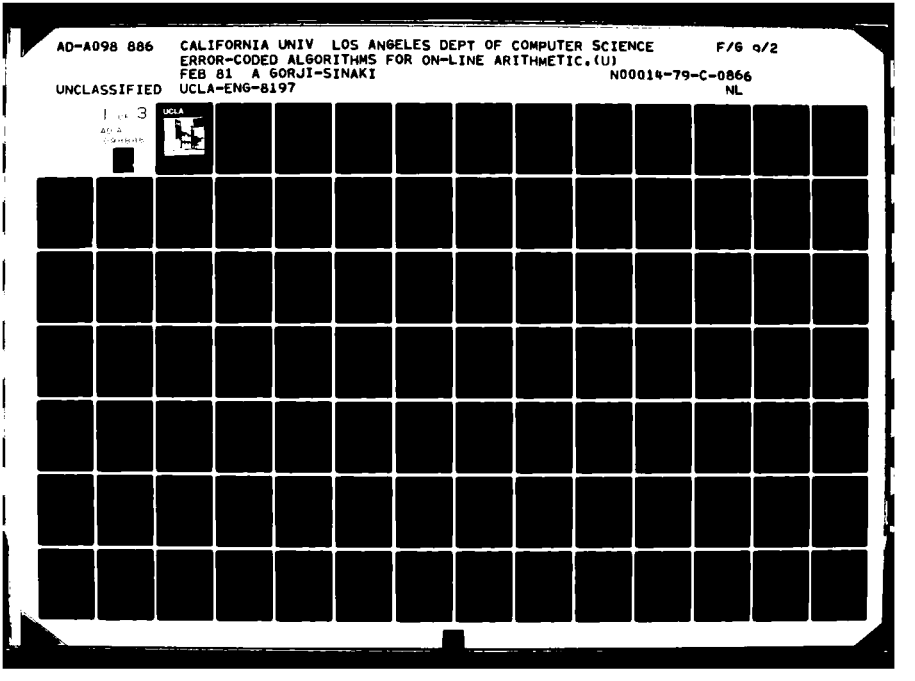
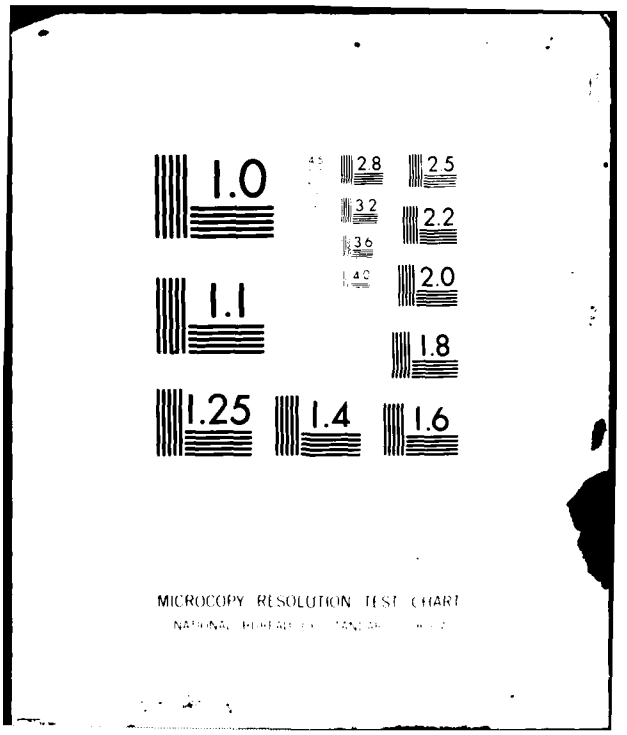
AD-A098 886 CALIFORNIA UNIV LOS ANGELES DEPT OF COMPUTER SCIENCE F/G 9/2
ERROR-CODED ALGORITHMS FOR ON-LINE ARITHMETIC.(U)
FEB 81 A GORJI-SINAKI N00014-79-C-0866
UNCLASSIFIED UCLA-ENG-8197 NL




1.0
2.8
2.5
3.2
2.2
3.6
4.0
2.0
1.1
1.8
1.25
1.4
1.6
MICROCOPY RESOLUTION TEST CHART

UCLA

LEVEL II

12

COMPUTER SCIENCE DEPARTMENT

AD A098886

DTIC FILE COPY

ERROR-CODED ALGORITHMS FOR ON-LINE ARITHMETIC

Abdolali Gorji-Sinaki

February 1981
Report No. CSD-810303

COMPUTER SCIENCE DEPARTMENT OFFICERS

Dr. Gerald Estrin, Chairperson
Dr. Bertram Bussell, Vice Chairperson
Mrs. Arlene C. Weber, Management Services Officer
Room 3731 Boelter Hall

COMPUTER SCIENCE DEPARTMENT RESEARCH LABORATORY

Dr. Leonard Kleinrock, Head
Room 3732 Boelter Hall

This report is part of a continuing series of technical reports initiated in January 1981 and is issued by the Computer Science Department Research Laboratory at UCLA. This technical report presents the latest research results established by the department members and its research staff. It is directed to the professional community and ranges from the presentation of short technical contributions to complete Ph.D. dissertations. Its function is to make available the latest results of our continuing research in a timely fashion. For a complete list of reports in this series you may contact The Department Archivist, at the address below.

University of California, Los Angeles
Computer Science Department
School of Engineering and Applied Science
3732 Boelter Hall
Los Angeles, California 90024

UNIVERSITY OF CALIFORNIA

Los Angeles

(6) Error-Coded Algorithms for On-Line Arithmetic.

(9) Final Rept.

A dissertation submitted in partial satisfaction of the requirements for the degree Doctor of Philosophy in Computer Science

(15) N00014-77-C-[illegible]

by

(10) Abdolali Gorji-Sinaki

(11) Feb 81 (12) 239

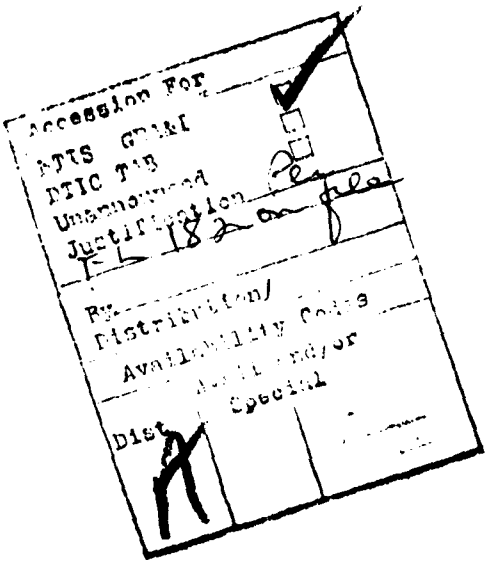
1981

DEDICATED TO MY PARENTS

PREFACE

The research described in this report, "Error-Coded Algorithms for On-Line Arithmetic," UCLA-ENG-8197, by Abdolali Gorji-Sinaki, was carried out as part of the Research in Distributed Processing, sponsored by the Office of Naval Research, Contract No. N00014-79-C-0866 under the direction of A. Avizienis, Principal Investigator, B. Bussell, M. Ercegovac, M. Gerla, S. Parker and D. Rennels, Co-Principal Investigators, in the Computer Science Department, School of Engineering and Applied Science, University of California, Los Angeles.

The research reported in this report was performed by the author under the guidance of his dissertation committee: Milos D. Ercegovac, Chairperson, Algirdas Avizienis, Bertram Bussell, Kirby Baker, and Clay Sprowls.



ACKNOWLEDGEMENTS

I would like to express my appreciation to my doctoral committee consisting of Professors Milos Ercegovac, Algirdas Avizienis, Bertram Bussell, Kirby Baker and Clay Sprowls. I owe special debt of gratitude to my Chairman, Dr. Milos Ercegovac, for his continued encouragement and guidance, his numerous suggestions and improvements, and above all his invaluable friendship during the course of this research. I am ever indebted to him. Special thanks also go to Dr. A. Grnarov for his remarks and particularly his much appreciated friendship.

I would also like to thank all my friends who helped me get through the difficult years of school. Particularly, I would like to thank J. Rahimian, N. Meshkati, F. Kianfar and Dr. F. Mohammadi for their constant encouragement and moral support throughout my graduate studies.

The support of the Tehran University of Technology during my studies at the University of California is sincerely appreciated. Gratitude is also owed to the Office of Naval Research for supporting this research, under the Contract Number N00014-79-C-0866.

Finally, I would like to offer my sincerest thanks to my parents and my brother CDR. Bagher Gorji-Ara for providing encouragement and support when it was most needed.

ABSTRACT OF THE DISSERTATION

Error-Coded Algorithms for On-Line Arithmetic

by

Abdolali Gorji-Sinaki
Doctor of Philosophy in Computer Science
University of California, Los Angeles, 1981
Professor Milos D. Ercegovac, Chair

Since on-line arithmetic requires relatively long sequences of operations in order to achieve speed-up over conventional arithmetic, it is important to protect on-line algorithms against hardware failures. If not protected, the hardware failures could quickly contaminate large number of results in progress due to tight coupling of the steps at the digit level. By detecting errors, as they occur, an effective, gracefully degradable organization could be achieved. Namely, error at any step of the algorithms would lead to restriction of precision (significance) of the

remaining steps but not catastrophic termination.

The main objective of this dissertation is to develop and demonstrate the feasibility of error-coded on-line arithmetic suitable for distributed systems.

In this thesis a set of error-coded on-line algorithms was developed for the four basic operations of addition/subtraction, multiplication and division. Low cost arithmetic error codes (Residue and AN Codes) were found to be suitable for this purpose.

Hardware design of the error-coded units at the gate level was considered. A residue-coded on-line division unit was designed based on a already designed digit-slice division unit.

A general mathematical model for the cost and speed of the error-coded units was derived and was compared with similar values when no error code is used. Finally, the effectiveness of the proposed detection/correction schemes was considered and proved.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1 Motivations and Objectives

This thesis is concerned with the development of a set of error coded basic algorithms for on-line arithmetic. In on-line processing the operands, as well as the results, flow through the arithmetic unit in a digit-by-digit manner starting with the most significant digit. On-line arithmetic provides a simple approach to achieve higher computational rates by allowing overlap at the digit level between the successive operations [ERC 75, TRI 77, IRW 77]. In particular, on-line arithmetic is highly attractive in some special applications, such as serial real-time processing, variable precision arithmetic and data flow architecture. Because of the serial nature of the algorithms, they might be used effectively in conjunction with large serial memories (CCDs, Bubble, etc.). On-line arithmetic offers a number of tradeoffs in system organization (interconnection and memory structures) that warrant additional research in this area.

Since on-line arithmetic requires relatively long sequences of operations in order to achieve speed-up over conventional arithmetic, it is important to protect on-line al-

gorithms against hardware failures. If not protected, the hardware failures could quickly contaminate large number of results in progress due to tight coupling of steps at the digit level. By detecting errors, as they occur, an effective, gracefully degradable organization could be achieved. Namely, error at the j-th step would lead to restriction of precision (significance) of the remaining steps but not catastrophic termination.

In this thesis we address the problem of developing such detection and correction procedures. We shall show that low-cost arithmetic error codes can be used effectively to support error-coded on-line arithmetic. Low cost error codes are advantageous because of the very simple checking procedure and cost-effective implementation.

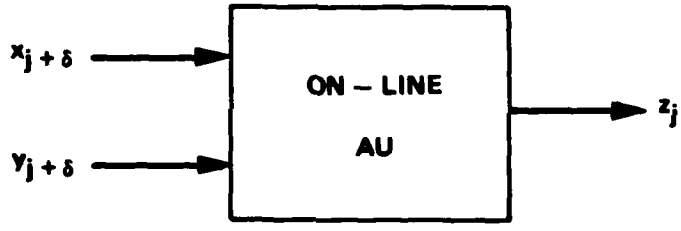
In the rest of the current chapter we review the state-of-art in on-line algorithms and consider some of their properties and applications. In Chapter 2 of this thesis a summary of the existing error-codes will be given. Chapter 3 and 4 are the main results of this work and deal with the presentation of the detection/correction schemes and their hardware implementation. In Chapter 5 performance of the error-coded units will be considered and their cost and speed will be compared with the corresponding ordinary on-line units. Chapter 6 contains the summary of the results obtained and some suggestions for the future research in the

area of on-line arithmetic.

1.2 On-Line Arithmetic

1.2.1 Definitions

By on-line algorithms we mean those arithmetic algorithms in which the operands as well as the results flow through the arithmetic unit in a digit-by-digit fashion, most significant digits first. These algorithms are such that, in order to generate the j-th digit of the result, $(j+\delta)$ digits of the corresponding operands are required. δ is called the on-line delay and is preferred to be as small as possible (Figure 1.1).



t :	1	2	...	δ	$\delta+1$	...	n	n+1	...	n+δ
INPUT :	x_1	x_2	...	x_δ	$x_{\delta+1}$	...	x_n	0	...	0
	y_1	y_2	...	y_δ	$y_{\delta+1}$	...	y_n	0	...	0
OUTPUT :	-	-	...	-	z_1	...	$z_{n-\delta}$		...	z_n

Figure (1.1)- An On-Line Arithmetic Unit

It is not difficult to see that the use of redundant number representation is mandatory for on-line algorithms. If we were to use a non-redundant number system, then even for

simple operations like addition and subtraction there is an on-line delay of δ=m due to carry propagation (m is the length of the operands). By using the signed-digit representation of numbers [AVI 61], it is possible to limit the carry propagation to one digit position.

1.2.2 Background

In general, an on-line algorithm is specified recursively in term of on-line representation of operands, results and some internal values. The following are the steps of a typical on-line algorithm:

1. Initialization:
2. Basic Recursion Step:

$$P_j = f(P_{j-1}, x_{j+\delta}, y_{j+\delta}, z_j)$$

Where f is a linear function and P_j is the partial result.

3. Selection Step:

$$z_{j+1} \leftarrow \text{SELECT}(P_j, x_{j+\delta}, y_{j+\delta})$$

Several of the well known basic algorithms satisfy the on-line property with respect to either the operands or the results. Consider, for example, conventional division which has the on-line property with respect to the quotient digits. Similarly, conventional multiplication has the on-line property with respect to the multiplier. This property has later been extended to the product digits as well.

As was mentioned earlier in this section, allowing redundancy in number representation will speed up the operation by limiting the carry propagation. A well known example is the totally parallel addition/subtraction with δ=1 [AVI 61]. More recent work in the area of on-line computation has been done by Ercegovac and Trivedi [ERC 75, TRI 77 and, TRI 78]. They developed on-line algorithms for multiplication and division. An overview of the generalized (with respect to radix) on-line algorithms for addition/subtraction, multiplication and, division has appeared in [IRW 77] along with the design of an on-line arithmetic unit. Others have extended on-line algorithms to encompass the on-line square rooting [ERC 78, OKL 78] and on-line normalization [GRN 79]. Also, a systematic method for derivation of on-line addition/subtraction, multiplication and, division algorithms appears in [GOR 80]. Several on-line algorithms such as y=ax+b, have been developed and used in iterative structures for array computations. Typical problems, such as matrix-vector multiplication and solving linear recurrence systems, have been investigated and corresponding solutions using on-line approaches are proposed and evaluated [ERC 80, GRN 80]. Other on-line algorithms and structures are reported in [CHU 80]. In order to efficiently explore and develop on-line algorithms a highly functional simulator has been developed and it is running on a DEC VAX II/780 [RAG 80].

1.2.3 Performance of On-Line Algorithms

There are seven major criteria that should be considered in evaluating the performance of a computational algorithm. These seven criteria are listed below:

1. Speed: throughput and delay
2. Cost: processor, processor types and, storage requirements.
3. Control efficiency
4. Interconnection requirements
5. Flexibility
6. Modularity
7. Reliability/"Robustness" of the algorithms.

The potential of on-line arithmetic in achieving a high performance has long been recognized and because of this property, a number of basic on-line algorithms have been developed in the literature (for the corresponding references see 1.2.2). Also a paper, written by Ercegovac and Grnarov [ERC 80], analyzes the performance of on-line arithmetic structures. It provides a relative comparison with the conventional arithmetic in computational problems such as the evaluation of scalar and vector expressions and recurrence systems. In what follows we analyze the performance of on-line algorithms with respect to the seven criteria mentioned above.

Speed

The speed-up of on-line algorithms is achieved by consistently applying a digit-serial mode of operation where the operands and the results are processed beginning with the most significant digit. Therefore, successive operations can be overlapped at the digit level and the interconnection requirements between arithmetic units are reduced to a minimum. Also by using a redundant representation of the partial results, it is possible to limit the carry propagation. Consequently the time required to compute one output digit can be made independent of the length of the operands.

Using a higher radix may also increase the speed of the computation by reducing the necessary number of steps for a given precision. But at the same time it increases the time to perform the basic recursion and the complexity of the corresponding on-line unit. Ercegovac and Grnarov in their paper [ERC 80] compared the speed of a multilevel on-line unit with the corresponding conventional unit demonstrating that for m=32 (m is the number of digits of the result), a network with two or more levels is faster in on-line arithmetic than in conventional arithmetic. They also showed that the time required to perform an operation is linearly proportional to the required precision. The results of their study indicate that by using on-line arithmetic (besides highly reduced communication requirements and modular, uni-

form implementation) one can expect an additional speed-up factor of 2-10.

Pipelining of successive operations can also be used as an effective speed-up technique [AVI 70, TUN 70, ERC 80]. In this scheme multiple on-line units are connected together in such a way that when the first unit completes processing, it passes all the necessary informations down the pipe and to the next unit. When one unit has completed all of the processing associated with the present operation, the next unit in line can begin generating the next result digit associated with that same instruction. In this way, the fraction arithmetic unit, which has been traditionally considered as a single stage of the pipeline, can be further decomposed into multiple stages to speed up processing even more. Chaining operations on result digit as they become available can increase processing speed even more.

Cost

The cost of on-line networks is a function of the cost of on-line arithmetic units and the cost of communication between the corresponding modules. Since in an on-line environment the interconnection between modules is via a one-digit wide link, the communication cost is obviously less than that of a conventional network. In a conventional network the number of data links between two modules is proportional to the number of digits transferred which is usually

a full precision number. On the other hand the number of modules required to implement a conventional arithmetic unit is at least proportional to m, while the corresponding number in an on-line environment is proportional to m/2 [ERC 80]. This factor also reduces the cost of on-line networks with respect to conventional one. Ercegovac and Grnarov [ERC 80] proved that the sufficient condition needed for an on-line, non-pipelined network to be less costly than the conventional one is that the cost of the on-line modules should not be more than twice the cost of the conventional module.

Control

Typically, the most random part of any system is its control logic. This randomness in logic makes the design of the control part of the system cumbersome and expensive. In order to alleviate this problem it is possible to microprogram the on-line unit possibly via a PLA to avoid randomness in control logic. On the other hand, since the basic computational step, in an on-line algorithm, is invariant at every step j and the only primitive arithmetic operation is addition, the control section can be designed in a straightforward manner. Ercegovac showed that the control requirements of an on-line unit is very simple. Assuming a synchronous mode of operation of the entire configuration, he showed that, the synchronizing clock pulses on which the transfer of digits occur, are all that is needed and the

same clock pulses, defining the basic step, are distributed to all units [ERC 75]. Finally, it is worth noting that, even though the on-line algorithms are iterative in nature, there are no convergence tests to be performed and this makes the control part simple and deterministic.

Interconnection Requirements

As was mentioned earlier in this section, one of the advantages of on-line units, in addition to a simple computing block, is the simplicity of communication between the corresponding modules. This reduction in internal and external communication requirements, comes from the fact that each module's control sees only its own state, therefore the interconnection among the elementary on-line units requires only single digit links. With regard to this, the structure using on-line arithmetic can be implemented in a highly modular manner. Pipelining of the on-line modules will also increase the complexity of the units, while the communication required between units will increase the links and therefore the pin count of each unit.

Flexibility

The ability of the on-line methods to perform without severe degradation while using the limited resources, (in other words their implementation flexibility) is also of practical importance. The on-line structures are easily ex-

tendible to accommodate either more levels or higher precision. Ercegovac proved that the proposed on-line method can be implemented under a wide range of speed/cost constraints in a simple way [ERC 75]. His method requires for the fastest evaluation, a configuration of m identical elementary units, but allows, in a straight forward manner, exploitation of its flexibility in tradeoff between the speed and cost. The cost change in precision, in the number of elementary units or in their complexity, affects the speed of computation linearly.

Modularity

It was previously mentioned that, the interconnections in an on-line arithmetic network are much simpler than in a conventional one, since only single digits are transferred between the operational units. Therefore, the structures using on-line arithmetic can be implemented in a highly modular manner. This property makes the arithmetic unit expandable both from the individual chip and the overall system viewpoint. In order to achieve this, the processing logic of on-line units should be partitioned to make it suitable to LSI. Logic partitioning involves the organization of the internal logic structures so that large functional areas(or arrays) on the chip can be grouped together and used repetitively. External to the chip, functional partitioning of the overall system requires a framework consisting of modules

which are completely self-contained processors, each having its own local store, processing logic, and the control necessary for the module to execute its function. Thus, each module acts as a small insular unit of logic. A good example of such a building block, for signed-digit arithmetic, is the single-package arithmetic processor called the Arithmetic Building Element (ABE) [AVI 70]. In the on-line environment, a typical module, implemented in a LSI technique, can be a 16 bit unit with a 4- operand adder, 4 registers, and a selection and carry block which can be by-passed so that a larger precision unit can be simply constructed by concatenating the required number of basic modules [ERC 75]. An organization of on-line unit as a linear array of identical modules operating in parallel is shown in Figure 1.2.

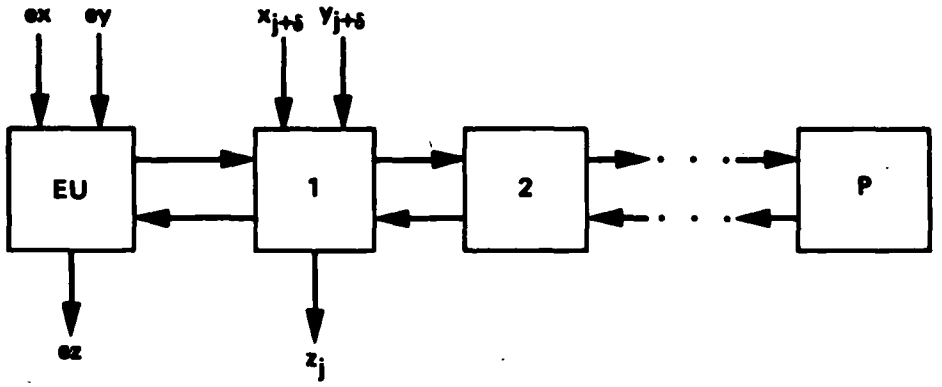


Figure (1.2) A Modular Organization of an On-Line Unit

Reliability

The reliability of on-line algorithms is of the major concern in this thesis. We are trying to enhance their "robustness" by applying error detection and correction to the already developed algorithms. The main result of this work is presented in Chapter 3 where error coded algorithms for four basic operations of addition, subtraction, multiplication and division are defined. Low-cost arithmetic error codes (Residue and AN codes) are found to be perfectly suitable for this purpose because the checking procedure is very simple and cost-effective to implement.

1.2.4 Applications

On-line arithmetic has a wide range of applications which makes it interesting for investigation in LSI technology. In what follows we review some of these applications with regard to the fact that some new applications may occur as advances are made in technology.

The most obvious use of on-line arithmetic is in the area of real-time processing in which the operands are generated serially by an analog-to-digital conversion process beginning with the most significant digits. An on-line unit can be used to process these digits as soon as they become available. This is unlike the conventional setup, where the processing unit must wait while the full precision operands are converted before starting the operation. The speed up benefits are obvious. In fact, any system designed to be of use in a real-time environment can make significant gains with the addition of an on-line module to its hardware.

Another possible application is in performing variable precision arithmetic. The existing algorithms and their simple implementation requirements are compatible with the required modularity of any variable precision unit. It is believed that sufficient register and adder widths can be provided by large scale integrated technology to provide enough "variable precision arithmetic" to meet the demands of most applications [AVI 62]. As a result, a unit which

operates in an on-line fashion can provide the ever popular microprocessor, a device traditionally restricted from most mathematical applications because of its short word length, with variable precision arithmetic capabilities.

Large-Scale computing applications of on-line arithmetic has been considered in [WAT 80]. In this research a multiprocessor organization for large-scale numerical behavior of algorithms has been studied.

On-line arithmetic can also be used in conjunction with large serial memories (CCDs, Bubble memories, etc.). This application depends on technological improvements of the foregoing memories. The major user of the large serial memories will be data base systems. Therefore, on-line arithmetic can provide instant processing capabilities for such a data base system.

As a final word, on-line arithmetic is complementary to other approaches that are used to achieve concurrency in execution of algorithms. For example, it can be used in minimal-depth tree-structured networks. In particular, the use of on-line arithmetic in non-linear recurrences systems would be advantageous [ERC 80]. They are very attractive in reconfigurable networks because of high modularity and simple interconnection.

CHAPTER 2

ERROR CODES

2.1 General Remarks on Error Codes

Computation without error remains an illusive goal of considerable importance in certain critical applications which require sophisticated and extensive computation with a high degree of system reliability. Recent advances in solid state technology have provided individual devices with exceptional reliability. In some systems, this improvement in device reliability has achieved sufficient systems reliability. However, in others, the large number of devices required has negated the improvement in reliability at the systems level. Such problems can be solved by the unlikely development of a perfect device which never fails. In the absence of such a device, one can expect greater use of the techniques of fault-tolerant computing to obtain improved systems reliability. Such improvements are not obtained without degradation in performance or increase in cost of the equipment, but in many applications, this tradeoff is justifiable.

One of the major approaches to fault-tolerant computing is the use of error detecting and error-correcting codes. In a practical system there are occasional errors, and it is the purpose of codes to detect and, perhaps, correct such errors. These codes cannot correct every conceivable pattern of errors but rather must be designed to correct only the most likely patterns. Much of coding theory has been based on the assumption that each symbol is affected independently, so that the probability of a given pattern depends only on the number of errors. For example, codes have been developed that correct any pattern of t or fewer errors in a block of n symbols. Also, for those systems in which errors may occur in bursts, some special kind of codes called "burst error codes" have been devised. In the following section we summarize the existing error codes with a special attention to arithmetic error codes. We will be using these types of codes throughout the rest of this dissertation.

2.2 Types of Codes

There are two fundamentally different types of codes: linear and non-linear codes. Between these two classes of codes, linear codes are more important and, because of this, have a well developed mathematical theory. In our review of error codes we only deal with a subset of all linear codes and therefore from now on we restrict our attention only to this class of codes. Linear codes are in turn divided into two classes: block codes and tree codes. The encoder of a block code breaks the continuous sequence of information digits into k-symbol sections or blocks. It then operates on these blocks independently according to the particular code to be employed. With each possible information block (k-symbols) is associated an n-tuple where n>k. The result, is now called a codeword. The quantity n is referred to as the code length or block length.

The other subset of linear codes, called a tree code, operates on the information sequence without breaking it up into independent blocks. Rather, the encoder for a tree code processes the information continuously and associates each long information sequence with a code sequence into l-symbol blocks, where l is usually a small number. Then, on the basis of this l-tuple and the preceding information symbols, it emits an m-symbol section of the code sequence. The name "tree code" stems from the fact that the encoding rules

for this type of code are most conveniently described by means of a tree graph.

Of the two classes of codes, the older block codes have a considerably better developed theory. The reason for this seems to be that block codes are more closely related to established, relatively well understood, mathematical structures. As a result, considerably more research has been done on them than on tree codes [PET 72]. Block codes are in turn divided into three basic subsets: Cyclic Codes, Non-cyclic Codes and Quasi-Cyclic Codes. Among these three categories we are interested in a subset of non-cyclic codes which are called "Arithmetic Error Codes". These codes differ from all those previously stated in that all operations are ordinary arithmetic. These codes are practical: they can be used for data transmission with encoding and operations performed by a general-purpose computer or they can be used to check the operation of an adder. There is an interesting similarity in structure between arithmetic codes and cyclic codes. Residue, Inverse-residue and AN codes belong to this class of codes. Figure 2.1 summarizes the relation among different error-codes in a hierarchical manner.

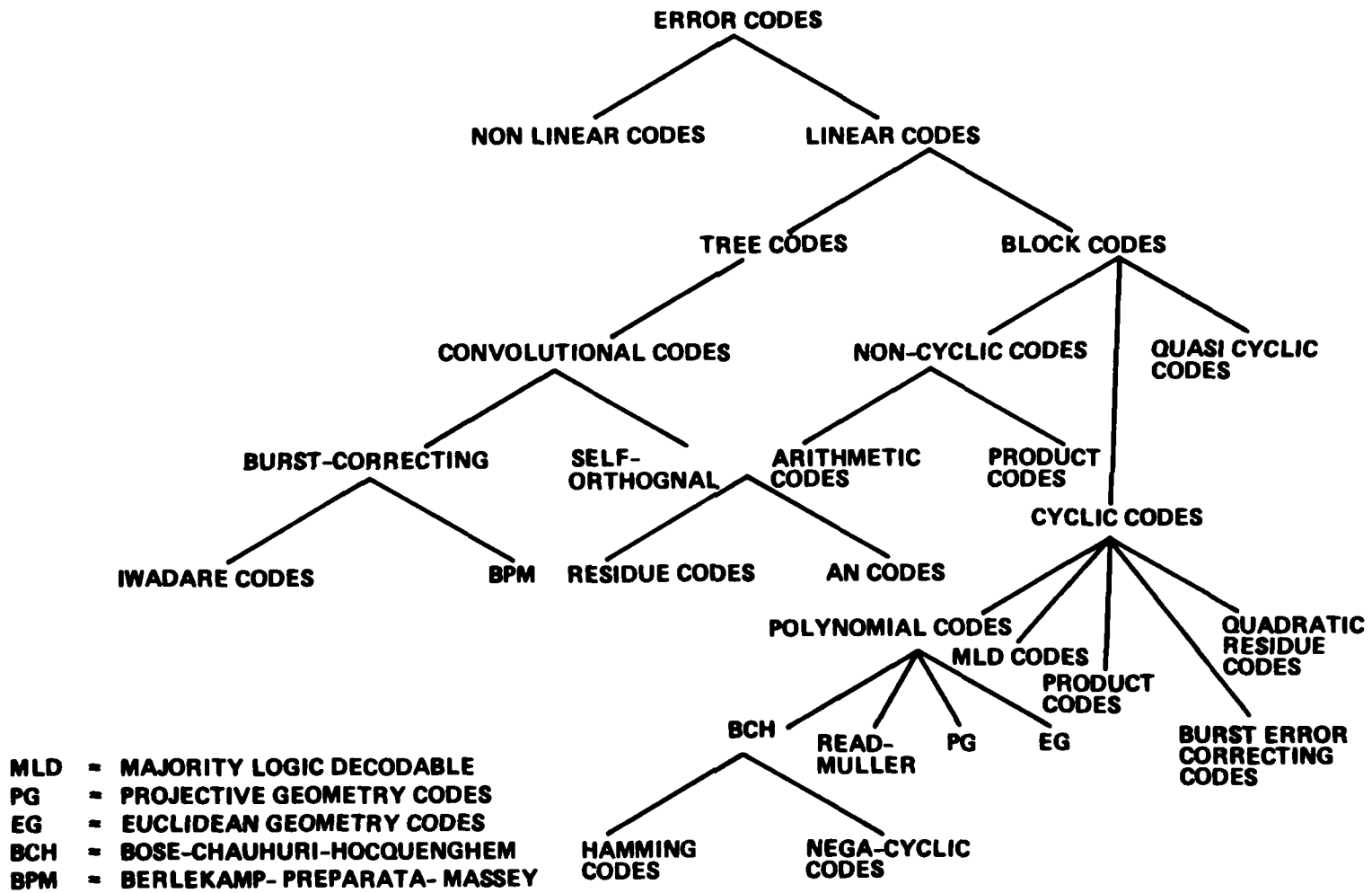


Figure 2.1 – A Summary of the Existing Error-Codes

2.2.1 Arithmetic Error Codes

An arithmetic code for us is a redundant representation of numbers having the property that certain errors can be detected and/or corrected in arithmetic operations using these codes. The representation is redundant in that the number of digits used for representing a number in a coded form may be larger than the minimum number of digits required if no error control is desired. The fundamental arithmetic operation is addition. Therefore, any useful arithmetic code must at least have the capability to check addition. Preferably, all other elementary operations, such as multiplication and division, should be checked as well.

To represent the set of integers $Z_m = \{0,1,....,m-1\}$, in the radix r system, the number of digits required, k, is the smallest integer greater than or equal to $\log_r m$. Instead of using k digits, as minimally required to represent Z_m , a redundant code uses n digits for some n>k. This may be in the nature of adding an extra n-k digits as checks to the non-redundant form of k digits; or it may be to denote each number $N \prec Z_m$ by a product AN for some constant integer A. Since these codes are used in checking arithmetic operations, it is important to define how these operations are carried out on redundant forms. Depending on how a number $N \prec Z_m$ is represented as an n-tuple or how arithmetic is performed on the codewords, the codes are classified as

separate or non-separate, and as systematic or non-systematic.

Definition 2.1

An arithmetic code, which has each codeword represented by, say, n digits is systematic if there exists a set of k digits (k<n) of the codeword representing the information and the remaining n-k digits representing the check(s).

A systematic code may treat the two parts, i.e., the information digits and the check digits, separately for the purpose of addition, thereby defining two or more independent addition structures, one for the information and the others for the checks; or it may treat each codeword as a single operand (or number) and define uniform addition rules for all n digits except perhaps for some end-around carries. A systematic code of the former type is called separate, and the latter none-separate. A similar division into separate and nonseparate classes can be made for all codes. Based on the preceding, arithmetic codes fall into these three major classes: 1) AN codes which are nonsystematic and therefore nonseparate; 2) separate codes with one or more residue check, for example (N, N mod A) residue code; and 3) the systematic subcodes, which are also called systematic non-separate codes. The AN codes were first introduced by Diamond [DIA 55], and their detection and error correction properties were discussed by Brown [BRO 60] and Peterson [PET

72]. The separate codes using a single residue check, such as (N, N mod A) code can only provide error detection for all arithmetic related operations, but not correction and, therefore, are of limited value [RAO 72]. In order to obtain error correction by use of separate codes, two or more residue checks are required, and that has led to the introduction of multiple residue codes [AVI 67, AVI 69, RAO 70]. The systematic subcodes appear to have error detection and/or correction properties similar to AN codes while preserving the advantages of systematic codes.

2.2.2 Low-Cost AN and Residue Codes

In an AN code, a given integer N is represented by the product A*N for some suitable constant A. A is commonly called the generator (and sometimes check modulus) of the code. The search for values of A which have a low-cost checking algorithm identified the class of low-cost arithmetic codes which employ the check moduli of the form

$$A=2^a-1, \text{ with integer } a>1. \qquad (2.1)$$

"a" is called the group length of the code [AVI 71]. AN codes with the check modulus 2^a-1 display an exceptional adaptability to binary arithmetic and have a low cost checking algorithm when the lengths of the operands are some multiple of the check length a.

As was mentioned earlier in this section, residue codes are categorized as separate codes. Indeed, Peterson proved that a separate code, meaning a code whose information and checks are separately processed, must be a residue code [PET 72]. The modulo A residue encoding for a number N attaches a check symbol C(N) to form a pair (N, C(N)). The value of C(N) is:

$$C(N)=N \bmod A \tag{2.2}$$

Where $(N \bmod A)=|N|_A$ designates the modulo A residue of N. The most significant differences of implementation between AN and residue codes are caused by the property of separateness. For residue codes, the operands N_1 and N_2 and their check symbols $C(N_1)$,$C(N_2)$ enter separate (main and check) processors which produce the main result N_3 and the check result $C(N_3)$. The checking algorithm computes ($N_3 \bmod A$) and compares it to $C(N_3)$. If the values are equal, either the correct result has been obtained, or a miss has occurred. Disagreement indicates a fault in either the main or the check processor. But for the nonseparate AN code the checking algorithm computes ($N_3 \bmod A$), where N_3 is the value of the result. The case ($N_3 \bmod A$)=0 indicates either a correct result or a miss. Note that the hardware cost of AN codes is caused by the greater complexity of the main processor, while for residue codes it is because of the need for a separate check processor. The error detection and correction properties of AN and residue codes are considered in

Chapter 5.

CHAPTER 3

ERROR CODED ON-LINE ALGORITHMS

In this chapter we present the main result of this thesis. Our goal is to develop a set of error coded basic algorithms for on-line arithmetic with the help of error codes we defined in Chapter 2 of this dissertation. As was mentioned in section 1.2.2, on-line algorithms for the four basic operations of addition/subtraction, multiplication and division have already been devised and the relevant results on this subject can be found in [ERC 75, TRI 77, TRI 78, IRW 77, GOR 80].

On-line algorithms have the property that if an error occurs at a certain step of an algorithm and if this error is detected immediately after generation and inhibited from spreading to the next module, then the operation of the following units can be continued although with less precision. Of course the final results have correspondingly less precision than the original operands. This shows that on-line algorithms have an intrinsic property of "graceful degradation". Of course, if there were some means of error correction, then this error would not affect the computation and there would not be any loss of precision. Our task is to

devise such algorithms with the capability of error detection and/or error correction.

In order to do this, we present two different schemes: 1)error detection with residue-coded operands, 2) error detection/correction with AN coded operands. Figure (3.1) is a general block diagram for an on-line unit with residue encoding.

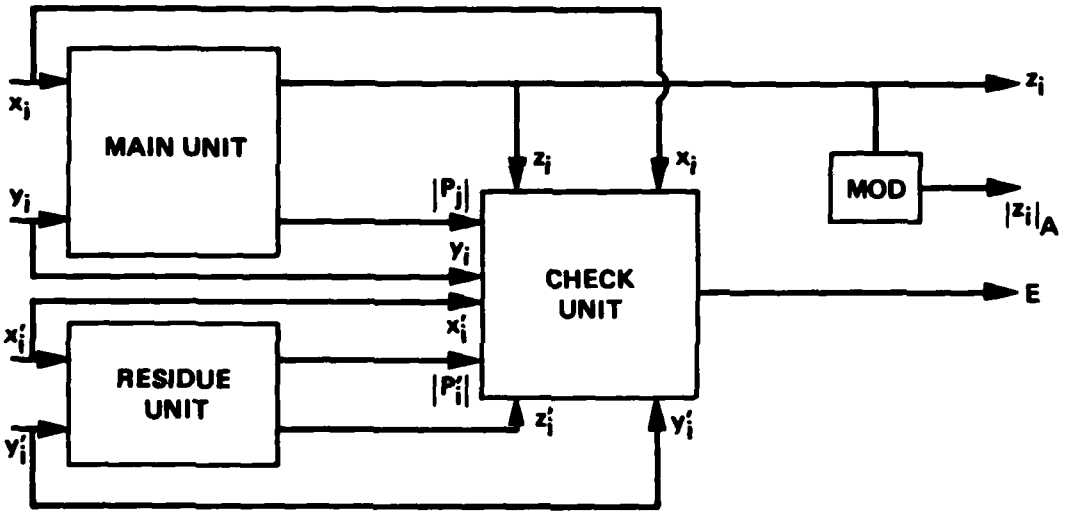


Figure (3.1)- Block Diagram of A Residue-Coded On-Line Unit

The operation of this unit is as follows:
Assume that the operands X and Y are represented by m digits in a radix r redundant number representation system, that is:

$$X = \sum_{i=1}^{m} x_i r^{-i} \qquad (3.1)$$

$$Y=\sum_{i=1}^{m} y_i r^{-i} \tag{3.2}$$

These two numbers flow through the MAIN unit digit-by-digit most significant digit first. The algorithm which is run by the MAIN unit (we call this algorithm "MAIN OP") is imposed on the incoming operands and after certain amount of delay, the result Z appears at the output, again digit-by-digit starting with MSD, such that:

$$Z=\sum_{i=1}^{m} z_i r^{-i} \tag{3.3}$$

At the same time the RESIDUE unit receives the residue of the corresponding digits of the MAIN unit. We represent this "residue" operands by X' and Y', such that:

$$X'=\sum_{i=1}^{m} x'_i r'^{-i} \tag{3.4}$$

$$Y'=\sum_{i=1}^{m} y'_i r'^{-i} \tag{3.5}$$

The following relation exists between these two sets of operands:

$$x'_i = x_i \bmod A \qquad \text{for } i=1,2,\ldots,m \tag{3.6}$$

$$y'_i = y_i \bmod A \qquad \text{for } i=1,2,\ldots,m \tag{3.7}$$

where A is the check modulus and was introduced in Chapter 2. We call the algorithm applied by RESIDUE unit as "RESIDUE OP". The output of the RESIDUE unit, with a similar manner, is represented by Z' and is:

$$Z' = \sum_{i=1}^{m} z'_i r'^{-i} \qquad (3.8)$$

Notice that the relation ($z'_i = z_i \bmod A$) is not necessarily satisfied.

After generation of z_i and z'_i, MAIN and RESIDUE units start working on the next set of inputs. At the same time z_i and z'_i along with some other information reach the CHECK unit. CHECK unit operates with an algorithm we call "DETECT OP". This unit after running the algorithm DETECT OP on z_i, z'_i and other received information, decides whether these results agree with each other or not. If the results do not agree then it sets an error flag which inhibits all the operations until the source of error is detected. For example, the current step can be repeated by the MAIN and RESIDUE units and if the error still persists, the operation can be continued with less precision. It is also possible to correct this error if we use biresidue codes instead of a single residue code [AVI 69, RAO 70]. In this thesis we do not address the problem of error correction by biresidue codes.

In the second scheme the operands X and Y are encoded with AN codes. Encoding is done by simply multiplying each digit of X and Y by a check modulus(A). Denote these encoded operands by X' and Y' respectively. Therefore:

$$X' = A * X$$

$$Y' = A * Y$$

such that:

$$x'_i = A * x_i \quad \text{for} \quad i = 1, 2, \ldots, m$$

$$y'_i = A * y_i \quad \text{for} \quad i = 1, 2, \ldots, m$$

The algorithm which operates of X' and Y' is the same as that for residue encoded operands (Algorithm MAIN OP). The output digit selection process in this algorithm should be such that the correct output digit (z'_i) is divisible by A. Therefore each single digit of the encoded operands and the results can be checked for divisibility by A. If any of these digits is not divisible by A, then it does not belong to the correct digit set and an error has occurred. The overall organization of the AN coded on-line unit is shown in Figure 3.2.

In this case we only need one MAIN Unit and the corresponding CHECK Unit which tests the operands and the results for divisibility by A.

This method has the following advantage over the residue encoding. If A is chosen appropriately then error correction is also possible in this case. We briefly mentioned that in order to correct single errors in the residue scheme we have to use biresidue codes instead of a single

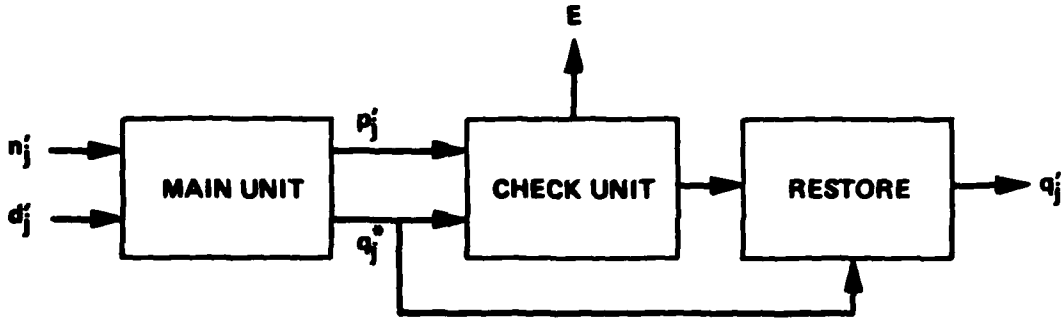


Figure (3.2)- Block Diagram of an AN-Coded On-Line Unit

residue code. Note that the hardware cost of AN codes is in the greater complexity of the MAIN processor, while for residue codes it is in the separate CHECK processor.

3.1 Error-Coded On-Line Division

3.1.1 Residue Coded Operands

Assume that the dividend N and the divisor D are represented by m digits in a radix-r redundant number system. Also assume that the residue of each digit with respect to a constant A ($A=2^a-1$) is attached to it and is transferred to the on-line DIVIDE unit. Therefore the coded operands are:

$$(N,N')=.(n_1,n'_1)(n_2,n'_2),.....,(n_m,n'_m)$$

$$(D,D')=.(d_1,d'_1)(d_2,d'_2),.....,(d_m,d'_m)$$

$$(Q,|Q|_A)=.(q_1,|q_1|_A)(q_2,|q_2|_A).....,(q_m,|q_m|_A)$$

n_i, d_i and q_i belong to the following symetric signed-digit sets [AVI 61]:

$$n_i \in \{-\rho'',...,-1,0,1,...,\rho''\} \quad (r-1)\geq\rho''\geq r/2 \quad (3.9)$$

$$d_i \in \{-\rho''',..,-1,0,1,..,\rho'''\} \quad (r-1)\geq\rho'''\geq r/2 \quad (3.10)$$

$$q_i \in \{-\rho,...,-1,0,1,...,\rho\} \quad (r-1)\geq \rho \geq r/2 \quad (3.11)$$

The algorithm "MAIN DIVIDE" which is run by the MAIN Unit is shown in the next page.

Algorithm "MAIN DIVIDE"

Step 1 [Initialization]:

$$P_0 = \sum_{i=1}^{\delta_{max}} n_i r^{-i}$$

$$D_0 = \sum_{i=1}^{\delta_{max}} d_i r^{-i}$$

$$Q_0 = 0$$

For $j=1,2,\ldots,m$ DO:

Step 2 [Selection]:

$$q_j = \text{SELECT}(rP_{j-1}, D_{j-1})$$

$$Q_j = Q_{j-1} + q_j r^{-j}$$

Step 3 [Input Digits]:

$$D_j = D_{j-1} + d_{j+\delta_{max}} r^{-j-\delta_{max}}$$

Step 4 [Basic Recursion]:

$$P_j = rP_{j-1} - q_j D_j + n_{j+\delta_{max}} r^{-\delta_{max}} - Q_{j-1} d_{j+\delta_{max}} r^{-\delta_{max}} \qquad (3.12)$$

Step 5 [End Do]

The algorithm run by the RESIDUE unit is similar to this and is named "RESIDUE DIVIDE".

Algorithm "RESIDUE DIVIDE"

Step 1 [Initialization]:

$$P'_0 = \sum_{i=1}^{\delta_{max}} n'_i r'^{-i}$$

$$D'_0 = \sum_{i=1}^{\delta_{max}} d'_i r'^{-i}$$

$$Q'_0 = 0$$

For j=1,2,....,m Do:

Step 2 [Selection]:

$$q'_j = \text{SELECT}(r'P'_{j-1}, D'_{j-1})$$

$$Q'_j = Q'_{j-1} + q'_j r'^{-j}$$

Step 3 [Input Digits]:

$$D'_j = D'_{j-1} + d'_{j+\delta_{max}} r'^{-j-\delta_{max}}$$

Step 4 [Basic Recursion]:

$$P'_j = r'P'_{j-1} - q'_j D'_j + n'_{j+\delta_{max}} r'^{-\delta_{max}}$$

$$- Q'_{j-1} d'_{j+\delta_{max}} r'^{-\delta_{max}} \qquad (3.13)$$

Step 5 [End Do]

r' is the radix of the RESIDUE unit which should be as small as possible. δ_{max} is the maximum of the on-line delays required by the MAIN and the RESIDUE Units and will be defined later. Also n'_i and d'_i are defined as was explained before:

$$n'_i = n_i \bmod A \qquad \text{for } i=1,2,\ldots,m$$

$$d'_i = d_i \bmod A \qquad \text{for } i=1,2,\ldots,m$$

Therefore,

$$n'_i, d'_i, |q_i| \in \{0,1,2,\ldots,(A-1)\} \tag{3.14}$$

The output of the RESIDUE Unit which is the quotient of residues are assumed to be in the following set:

$$q'_i \in \{-\rho', \ldots, -1, 0, 1, \ldots, \rho'\} \quad (r'-1) \geq \rho' \geq r'/2 \tag{3.15}$$

In what follows we prove that the Algorithm "MAIN DIVIDE" and similarly the Algorithm "RESIDUE DIVIDE" converge to the correct value of the quotient.

Proof of Convergence

By induction on j in the basic recursion formula(Eq. 3.12) we get:

$$j=1 \rightarrow P_1 = r \sum_{i=1}^{\delta_{max}} n_i r^{-i} - q_1 \sum_{i=1}^{1+\delta_{max}} d_i r^{-i} + n_{1+\delta_{max}} r^{-\delta_{max}}$$

$$= r \sum_{i=1}^{1+\delta_{max}} n_i r^{-i} - q_1 \sum_{i=1}^{1+\delta_{max}} d_i r^{-i}$$

$$j=2 \rightarrow P_2 = r^2 \sum_{i=1}^{1+\delta_{max}} n_i r^{-i} - rq_1 \sum_{i=1}^{1+\delta_{max}} d_i r^{-i} - q_2 \sum_{i=1}^{2+\delta_{max}} d_i r^{-i}$$

$$+ n_{2+\delta_{max}} r^{-\delta_{max}} - q_1 r^{-1} d_{2+\delta_{max}} r^{-\delta_{max}}$$

$$= r^2 \sum_{i=1}^{2+\delta_{max}} n_i r^{-i} - (rq_1+q_2) \sum_{i=1}^{2+\delta_{max}} d_i r^{-i}$$

Continuing this procedure we obtain P_j as follows:

$$P_j = r^j \sum_{i=1}^{j+\delta_{max}} n_i r^{-i} - r^j \left[\sum_{i=1}^{j} q_i r^{-i} \right] \left[\sum_{i=1}^{j+\delta_{max}} d_i r^{-i} \right] \quad (3.16)$$

If j=m then:

$$P_m = r^m \sum_{i=1}^{m} n_i r^{-i} - r^m \left[\sum_{i=1}^{m} q_i r^{-i} \right] \left[\sum_{i=1}^{m} d_i r^{-i} \right]$$

or

$$r^{-m} P_m = N - Q*D$$

From this equation Q is obtained:

$$Q = \frac{N}{D} - r^{-m} \frac{P_m}{D} \quad (3.17)$$

Therefore, by devising a quotient digit selection procedure, SELECT in step 4 of the Algorithm "MAIN DIVIDE" such that

$$|P_m| < D$$

the quotient $Q = \frac{N}{D}$ can be computed to m digits of precision.

A similar proof is valid for the RESIDUE Unit:

$$P'_j = r'^j \sum_{i=1}^{j+\delta_{max}} n'_i r'^{-i} - r'^j \left[\sum_{i=1}^{j} q'_i r'^{-i}\right] * \left[\sum_{i=1}^{j+\delta_{max}} d'_i r'^{-i}\right] \quad (3.18)$$

$$Q' = \frac{N'}{D'} - r'^{-m}\frac{P'_m}{D'} \quad (3.19)$$

and assuming $|P'_m| < D'$ then $Q' = \frac{N'}{D'}$ to m digits of precision.

3.1.1.1 The Error Detection Algorithm

The purpose of this section is to find an algorithm that can detect an error at each step of the on-line division process. This algorithm is run after generation of q_i and q'_i by the MAIN and RESIDUE Units and will determine whether these quotient digits are correct or not. If an error is detected then the current step is repeated, otherwise the division process proceeds as usual.

From Equations (3.16) and (3.18) we have:

$$\left[\sum_{i=1}^{j} q_i r^{-i}\right]\left[\sum_{i=1}^{j+\delta_{max}} d_i r^{-i}\right] = \sum_{i=1}^{j+\delta_{max}} n_i r^{-i} - r^{-j} P_j$$

and

$$\left[\sum_{i=1}^{j} q'_i r'^{-i}\right]\left[\sum_{i=1}^{j+\delta_{max}} d'_i r'^{-i}\right] = \sum_{i=1}^{j+\delta_{max}} n'_i r'^{-i} - r'^{-j} P'_j$$

By dividing these two equations and getting the residues of

both sides and also assuming:

$$|r|_A=|r'|_\lambda=1 \tag{3.20}$$

we get the following equation:

$$\left|\left|\left|\sum_{i=1}^{j} q_i\right|*\left|\sum_{i=1}^{j+\delta_{max}} n'_i\right|-|P'_j|\right|\right|=$$

$$\left|\left|\left|\sum_{i=1}^{j} q'_i\right|*\left|\sum_{i=1}^{j+\delta_{max}} n_i\right|-|P_j|\right|\right| \tag{3.21}$$

In the equation above:

$$|x|=|x|_A=x \bmod A$$

To simplify Eq. (3.21) the following change of parameters are done:

$$\left|\sum_{i=1}^{j} q_i\right|_A=X_j \quad \text{and} \quad \left|\sum_{i=1}^{j} q'_i\right|_A=X'_j$$

$$\left|\sum_{i=1}^{j+\delta_{max}} n'_i\right|_\lambda=Y'_j \quad \text{and} \quad \left|\sum_{i=1}^{j+\delta_{max}} n_i\right|_\lambda=Y_j$$

Therefore (3.21) becomes:

$$\left|\left|X_j\right|*\left|Y'_j\right|-\left|P'_j\right|\right|_A=\left|\left|X'_j\right|*\left|Y_j\right|-\left|P_j\right|\right|_A \tag{3.22}$$

The correctness of this equation is the test that we perform to detect an error in the division process. The following algorithm, run by the CHECK Unit, performs this test.

Algorithm "DETECT DIVIDE"

Step 1 [Initialization]:

$$X_0=X'_0=0$$

$$Y_0=\left|\sum_{i=1}^{\delta_{max}} n_i\right|_A$$

$$Y'_0=\left|\sum_{i=1}^{\delta_{max}} n'_i\right|_A$$

For j=1,2,....,m Do:

Step 2 [Input Digits]:

$$X_j=|X_{j-1}+q_j|_A$$

$$X'_j=|X'_{j-1}+q'_j|_A$$

$$Y_j=|Y_{j-1}+n_j+\delta_{max}|_A$$

$$Y'_j=|Y'_{j-1}+n'_j+\delta_{max}|_A$$

P_j and P'_j

Step 3 [Check for Error]:

$$Z_j=|X_j*(Y'_j-|P'_j|_A)|_A$$

$$Z'_j=|X'_j*(Y_j-|P_j|_A)|_A$$

If $(Z_j \neq Z'_j)$ E=1, GOTO ERROR SUBROUTINE

Step 4 [End Do]

3.1.1.2 Determination of δ_{max}

In order to have overlap between the adjacent selection regions of the P_j-D_j plot, the minimum index difference (δ) for the case of redundant dividend and divisor is found to be [GOR 80]:

$$\delta=\left\lceil 2+\log_r \frac{2(k''+kk''')}{(2k-1)(1-k''')} \right\rceil \tag{3.23}$$

k, k'' and k''' are defined as:

$$k=\frac{\rho}{r-1}$$

$$k''=\frac{\rho''}{r-1}$$

$$k'''=\frac{\rho'''}{r-1}$$

Since division in the RESIDUE Unit is performed with non-redundant operands, we get [GOR 80]:

$$\delta'=\left\lceil 2-\log_{r'} \frac{2k'-1}{k'+1} \right\rceil \tag{3.24}$$

where $k'=\frac{\rho'}{r'-1}$. We define δ_{max} to be the maximum of δ and δ'.

$$\delta_{max}=MAX(\delta,\delta') \tag{3.25}$$

3.1.1.3 Radix of The RESIDUE Unit

As was mentioned earlier, the radix of the RESIDUE Unit is an important factor in the design of the error-coded units. Because, as r' increases the amount of hardware needed

for the RESIDUE Unit increases. In the extreme case where r=r' then the detection process is merely duplication of the MAIN Unit. On the other hand there are some lower bounds for r' that should be met. These bounds are calculated as follows:

Since residue digits (n'_i, d'_i) are assumed to be in radix r' number system we have:

$$n'_i, d'_i \leq r'-1$$

using Eq. (3.14) we get:

$$A-1 \leq r'-1 \text{ or } r' \geq A \tag{3.26}$$

from (3.20) and (3.26) we obtain:

$$r'=M'A+1 \quad \text{for} \quad M'=1,2,.... \tag{3.27}$$

and if we assume that A is a low-cost modulus $(A=2^{\alpha}-1)$ then:

$$r'=M'2^{\alpha}-M'+1 \quad \text{for} \quad M'=1,2,... \tag{3.28}$$

also from (3.20) we get:

$$r=MA+1 \quad \text{for} \quad M=1,2,... \tag{3.29}$$

3.1.1.4 An Example of The Error Detection Process

The following is a numerical example of the error-detection process when residue-coded operands are used.

Assume:

$$r=10, k=k''=k'''=\frac{5}{9}$$

$$r'=4, k'=\frac{2}{3}$$

$$A=3$$

using (3.23) and (3.24) we get $\delta=4$ and $\delta'=4$ respectively. Therefore:

$$\begin{cases} n_i, d_i, q_i \in \{\bar{5}, \bar{4}, \ldots, \bar{1}, 0, 1, \ldots 4, 5\} \\ n'_i, d'_i \in \{0, 1, 2\} \quad \text{for} \quad i=1, 2, \ldots, m \\ q'_i \in \{\bar{2}, \bar{1}, 0, 1, 2\} \end{cases}$$

Assume:

$$D=.550\overline{23}4$$

and

$$N=.13\overline{34}01$$

Therefore:

$$D'=.220101$$

$$N'=.100201$$

using [GOR 80] we get the following rP_j-D_j plot for selection of the quotient digits of the MAIN Unit.

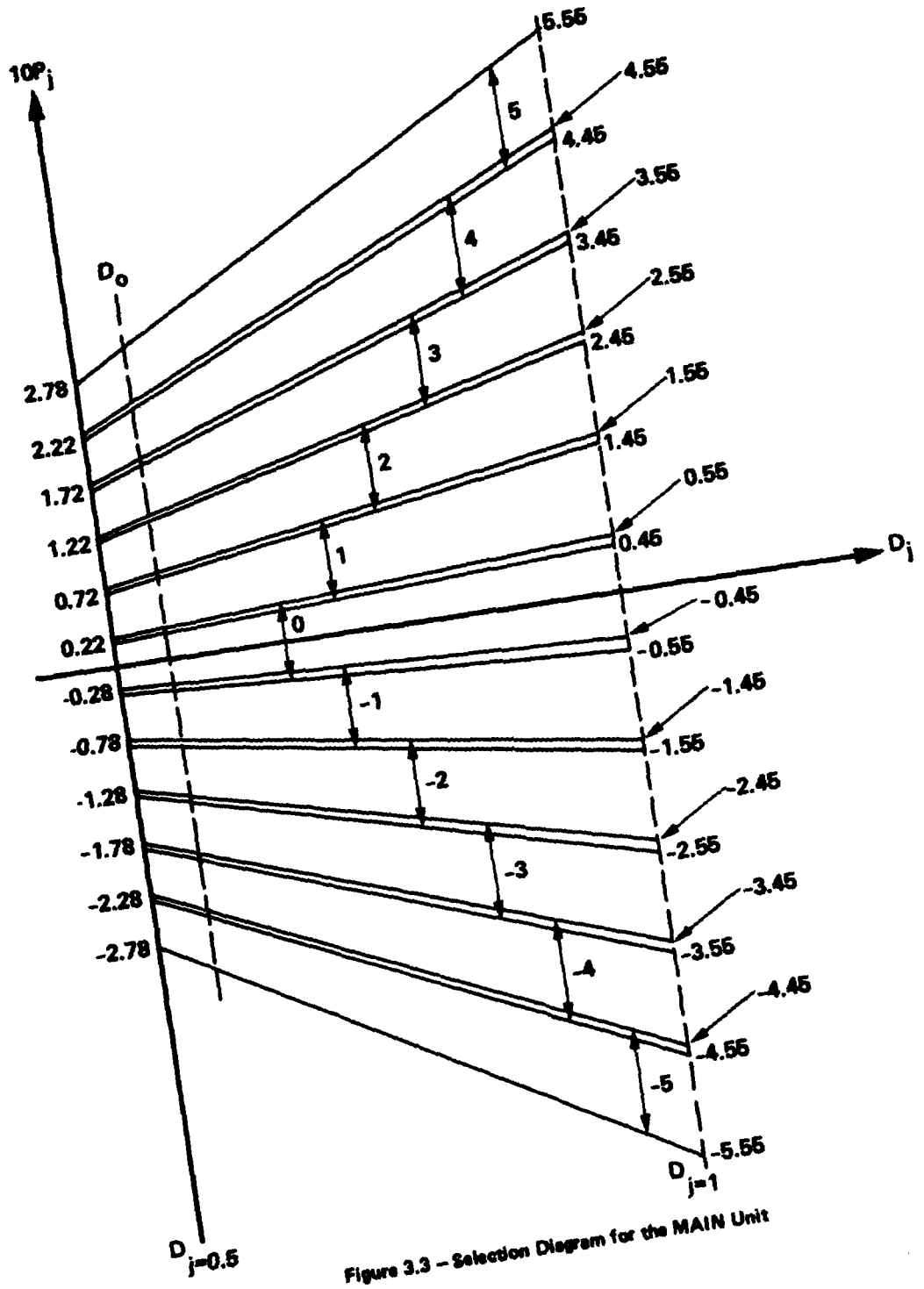


Figure 3.3 – Selection Diagram for the MAIN Unit

Using the selection diagram shown in Figure (3.3) and the Algorithm MAIN DIVIDE, the following table is obtained for the operation of the MAIN Unit (Table 3.1).

j	P_{j-1}	Q_j	D_j	q_j
1	.1266	.2	.54977	2
2	.16646	.23	.549774	3
3	.015298	.230	.549774	0
4	.15298	.2303	.549774	3
5	-.119522	$.2303\overline{2}$	.549774	$\overline{2}$
6	-.095672	$.230\overline{32}$	.549774	$\overline{2}$

Table (3.1)- Results Obtained by the MAIN Unit (EX. 3.1.1.4)

According to this table:

$$Q=Q_6=.230\overline{32}$$

Figure (3.4) shows the selection diagram for the RESIDUE Unit.

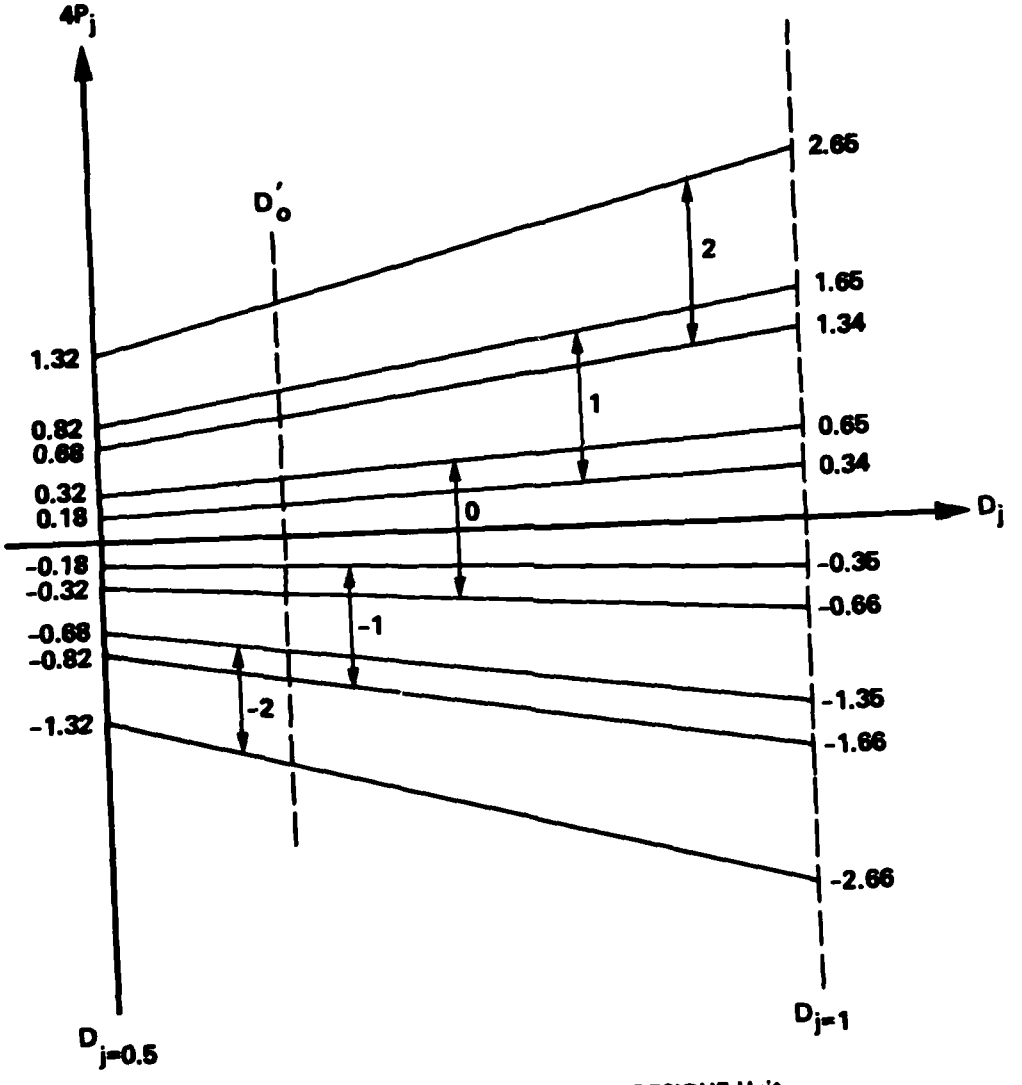


Figure 3.4 – Selection Diagram for the RESIDUE Unit

Similar to Table (3.1), Table (3.2) shows the results obtained by the RESIDUE Unit.

j	P'_{j-1}	Q'_j	D'_j	q'_j
1	.1002	.2	.2201	2
2	-.0322	$.2\bar{1}$	.2201	$\bar{1}$
3	-.101213	$.2\bar{1}\bar{2}$	.2201	$\bar{2}$
4	.022012	$.2\bar{1}21$	.2201	1
5	.000013	$.2\bar{1}210$	.2201	0
6	.00013	$.2\bar{1}2100$	.2201	0

Table (3.2)- Results Obtained by the RESIDUE Unit

According to Table (3.2):

$$Q'=Q'_6=(.2\bar{1}2100)_4$$

The information needed by the CHECK Unit at the j-th step of the algorithm are n_j, n'_j, q_j , $q'_j, |P_j|_A$ and $|P'_j|_A$. Table (3.3) is obtained by using the Algorithm DETECT DIVIDE and summarizes the operation of the CHECK Unit for this example.

Since $Z_j=Z'_j$ for $j=1,2,\ldots,6$ then all the operations have been correctly performed or a miss has occurred.

j	X_j	X'_j	Y_j	Y'_j	Z_j	Z'_j	$Z_j=Z'_j$
1	2	2	0	0	2	2	check
2	2	1	1	1	0	0	check
3	2	2	1	1	0	0	check
4	2	0	1	1	0	0	check
5	0	0	1	1	0	0	check
6	1	0	1	1	0	0	check

Table (3.3)- Results Obtained by the CHECK Unit.

Now assume that at step 3 of the 'MAIN DIVIDE' Algorithm an error in the Multi-Input Redundant Adder causes the partial remainder P_3 (=0.15298) to be incorrect. Assume this wrong result (P_3^*) is:

$$P_3^*=0.15296$$

Continuing the algorithm "DETECT DIVIDE" from step 3 we get:

j=3

$$X_3=|2+0|_A=2$$

$$X'_3=|1-2|_A=2$$

$$Y_3=|1+0|_A=1$$

$$Y'_3 = |1+0|_A = 1$$

$$Z_3 = |2*(1-1)|_A = 0$$

$$Z'_3 = |2*(1-2)|_A = 1$$

Since $Z_3 \neq Z'_3$ this error will be detected by the CHECK Unit.

When no error is detected by the CHECK unit, the current quotient q_j is delivered to the next on-line unit along with its residue modulo A [notice that (q_j mod A) is not necessarily equal to q'_j]. These two constitute one of the operands of the following on-line unit.

3.1.2 AN Coded Operands

In the previous section we discussed the detection of errors in an on-line divide unit when the dividend and the divisor are residue coded. As was mentioned before, it is possible to use AN coded operands for the purpose of error detection and/or error correction. In this section we present a summary of the proposed algorithms when AN coded operands are used.

Again we denote the operands by N, D and Q for the dividend, divisor and the quotient. Encoded operands are obtained by simply multiplying each digit of the N, D and Q by the check modulus A. Denote these encoded operands by N', D' and Q'. The table of the next page shows the correspondence between two sets of operands and the results. For the reason which will be explained later, the digits of the dividend (N) are multiplied by A^2 instead of A [AVI 73].

$$N=\sum_{i=1}^{m} n_i r^{-i} \quad \text{and} \quad N'=\sum_{i=1}^{m} n'_i r^{-i}$$

$$D=\sum_{i=1}^{m} d_i r^{-i} \quad \text{and} \quad D'=\sum_{i=1}^{m} d'_i r^{-i}$$

$$Q=\sum_{i=1}^{m} q_i r^{-i} \quad \text{and} \quad Q'=\sum_{i=1}^{m} q'_i r^{-i}$$

$$q_j \in \{-\rho,..,0,..,\rho\} \quad \text{and} \quad q'_j \in \{-A\rho,..,0,..,A\rho\} \tag{3.30}$$

$$\begin{cases} n_j \in \{-\rho'',\ldots,\bar{1},0,1,\ldots,\rho''\} \text{ and} \\ n'_j \in \{-A^2\rho'',\ldots,-A^2,0,A^2,\ldots,A^2\rho''\} \end{cases} \tag{3.31}$$

$$\begin{cases} d_j \in \{-\rho',\ldots,\bar{1},0,1,\ldots,\rho'\} \text{ and} \\ d'_j \in \{-A\rho',\ldots,\bar{A},0,A,\ldots,A\rho'\} \end{cases} \tag{3.32}$$

$$k' > D \geq \frac{1-k'(1-r^{-m+1})}{r} \text{ and } Ak' > D' \geq A\frac{1-k'(1-r^{-m+1})}{r} \tag{3.33}$$

$$k \geq Q_j \geq -k \quad \text{and} \quad Ak \geq Q'_j \geq -Ak \tag{3.34}$$

such that:

$$\begin{cases} n'_j = A^2 n_j \\ d'_j = A d_j \quad \text{for} \quad j=1,2,\ldots,m \\ q'_j = A q_j \end{cases} \tag{3.35}$$

The division algorithm which operates on encoded operands is exactly same as "MAIN DIVIDE" Algorithm discussed in the previous section. Also the proof of the convergence of the algorithm in this case is similar to what was mentioned before. Following the same procedure we get:

$$P'_m = r^m[N' - Q'D'] \text{ or } Q' = \frac{N'}{D'} - r^{-m}\frac{P'_m}{D'}$$

Now if we prove that

$$|P'_m| < AD' \tag{3.36}$$

The quotient $Q' = \frac{N'}{D'}$ to m digits of precision.

$$Q' = \frac{N'}{D'} = \frac{A^2N}{AD} = A\frac{N}{D} = AQ \tag{3.37}$$

which is the correct result and shows the reason why we have to multiply n_j by A^2 instead of A.

3.1.2.1 Selection of The Quotient Digits

One of the most important factors in the design of the AN-coded division unit is the selection procedure. As was mentioned before, selection is such that the correct quotient is a multiple of the check modulus(A). With the help of the basic recursion formula (3.12) and following the procedure given in [GOR 80] the bounds on partial remainder are obtained:

$$kAD'_j - A^2(k'' + kk')r^{-\delta} \geq P'_j \geq -kAD'_j + A^2(k'' + kk')r^{-\delta} \tag{3.38}$$

By letting j=m in (3.38) we get:

$$AD' \geq |P'_m| \geq -AD'$$

Therefore, Eq. (3.36) is satisfied and Q' is indeed the correct quotient up to m digits. Also following the procedure given in [GOR 80] we get the following set of selection equations:

$$(i'+Ak)D'_j - A^2(k''+kk')r^{-\delta+1} \geq rP'_j \geq (i'-Ak)D'_j + A^2(k''+kk')r^{-\delta+1} \quad (3.39)$$

This condition can be graphically described by means of a P'_j-D'_j plot [ATK 68]. It consists of a family of curves which are linear function of D'_j with q'_j as parameter ranging from $-A\rho$ to $+A\rho$ in steps of A. The area between maximum rP'_j and the minimum rP'_j will be denoted the $q'_j=i'$ region. A given value of D'_j and rP'_j will correspond to a point in an i'- selection region. The quotient digit q'_j is, therefore, i' and is used in forming the next partial remainder. Figure (3.5) is an example of a full P'_j-D'_j plot with r=2, $k=k'=k''=1$, A=3 and $\delta=4$.

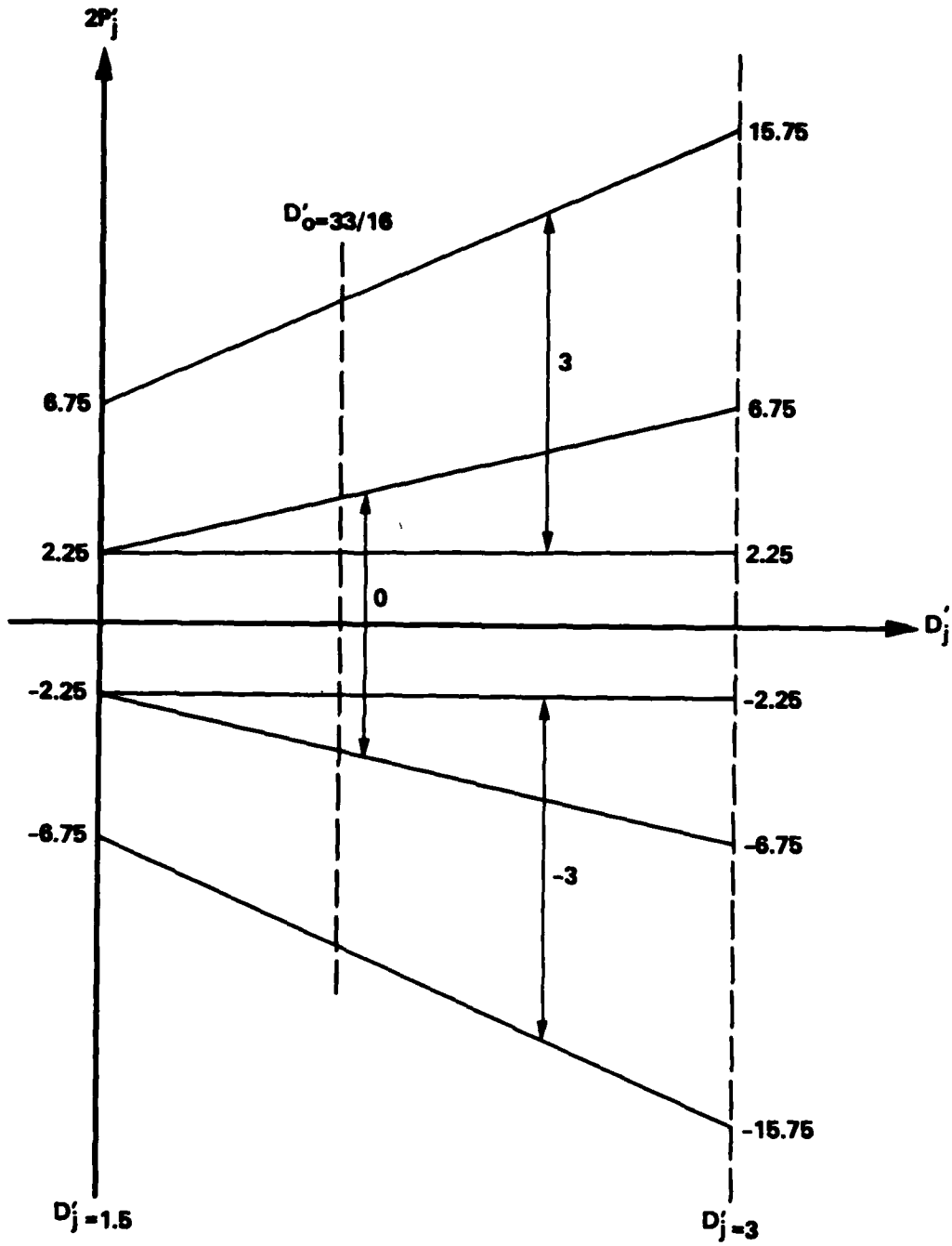


Figure 3.5 – An Example of A $P'_j - D'_j$ Plot

3.1.2.2 Determining The Minimum Index Difference

The minimum allowable value of δ can be determined by requiring that the lower bound of a $q'_j=i'$ selection region and the upper bound for the corresponding $q'_j=i'-A$ selection regions intersect at the minimum value of D'. Therefore, by using Eq. (3.39) we get:

$$r^{-\delta+1} \leq \frac{2k-1}{2A(k'' + kk')} (D'_j)_{min} \tag{3.40}$$

From Eq. (3.33) assuming $m \rightarrow \infty$ minimum value of D'_j is found to be:

$$(D'_j)_{min} = A\frac{1-k'}{r}$$

inserting this into Eq. (3.40) the worst case δ is found:

$$\delta \geq 2+\log_r \frac{2(k'' + kk')}{(2k-1)(1-k')} \tag{3.41}$$

By referring to [GOR 80] we find that this δ is exactly the same as that found for ordinary operands. Therefore the proposed encoding does not change the minimum delay required.

3.1.2.3 An Example of Division With AN-Coded Operands

Assume:

$$r=2,\ k=k'=k''=1 \text{ and } A=3$$

Also we assume that D is normalized (not pseudonormalized). Therefore:

$$(D'_j)_{min}=1.5$$

inserting this in (3.40) results in $\delta \geq 4$. Assume:

$$\delta=4$$

from Equation (3.30), (3.31) and (3.32) we get:

$$\begin{cases} q'_j \prec \{\bar{3},0,3\} \\ n'_j \prec \{\bar{9},0,9\} \\ d'_j \prec \{\bar{3},0,3\} \end{cases}$$

and also:

$$\begin{cases} 3 > D' \geq \frac{3}{2} \\ 3 \geq Q'_j \geq -3 \\ 9 > N' > -9 \end{cases}$$

Plugging the given values in Eq. (3.39) the selection regions are obtained:

$$(i'+3)D'_j-18*2^{-3} \geq (2P'_j)^{q'_j=i'} \geq (i'-3)D'_j+18*2^{-3}$$

where $i' \prec \{\bar{3},0,3\}$

Figure (3.5) shows the selection regions obtained from this set of inequalities. Now assume the dividend and the divisor are:

$$N'=.9\bar{9}099\bar{9}$$

$$D'=.33\bar{3}303$$

Following the "MAIN DIVIDE" algorithm shown in Section 3.1.1 of this thesis, Table (3.4) will result.

j	P'_{j-1}	Q'_j	D'_j	q'_j
1	$.9\bar{9}09$	.3	$.33\bar{3}30$	3
2	.0	.30	$.33\bar{3}303$	0
3	$.00\bar{99}$	.300	$.33\bar{3}303$	0
4	$.00\bar{99}$	$.300\bar{3}$	$.33\bar{3}303$	$\bar{3}$
5	.090909	$.300\bar{3}3$	$.33\bar{3}303$	3
6	$.000\bar{99}$	$.300\bar{3}30$	$.33\bar{3}303$	0
7	$.00\bar{99}$	-	-	-

Table (3.4)- An Example of AN-Coded Division

According to this table:

$$Q'=Q'_6=(300\bar{3}30)_2$$

By looking at columns two and three of the above table, it can be confirmed that all the digits of P'_{j-1} and Q'_j are multiples of the check modulus (A=3). Therefore, the necessary condition for the correctness of the division process is satisfied.

3.2 Error-Coded On-Line Multiplication

3.2.1 Residue Coded Operands

Assume that the multiplicand (X) and the multiplier (Y) are represented by m digits in a radix-r redundant number system. Also assume that the residue of each digit with respect to a constant A ($A=2^a-1$) is attached to it and is sent to the on-line multiplication unit. Therefore the coded operands are:

$$(X,X')=(x_1,x'_1)(x_2,x'_2)\ldots(x_m,x'_m)$$

$$(Y,Y')=(y_1,y'_1)(y_2,y'_2)\ldots(y_m,y'_m)$$

The product R is also represented by an m digit radix-r redundant number. The residue of each product digit is also attached to it while leaving the multiplication unit.

$$(R,|R|_A)=(p_1,|p_1|_A)(p_2,|p_2|_A)\ldots(p_m,|p_m|_A)$$

Since X and Y are assumed to be redundant, x_i and y_i belong to the following digit set:

$$x_i,\ y_i \in \{-\rho'',\ldots,\bar{1},0,1,\ldots,\rho''\} \qquad (3.42)$$

X' and Y' are not redundant, therefore:

$$x'_i,\ y'_i,\ |p_i|_A \in \{0,1,\ldots,(A-1)\} \qquad (3.43)$$

Relation (3.43) is obtained from the definition of the residue function.

X and Y are assumed to be bounded by a positive constant M such that:

$$M \geq X,\ Y \geq -M \tag{3.44}$$

and similarly:

$$M' \geq X',\ Y' \geq 0 \tag{3.45}$$

The operands pass through a MAIN Unit which performs the algorithm "MULT" given in Appendix C. The result R is also in a redundant number system such that:

$$R = \sum_{i=1}^{m} p_i r^{-i}$$

and p_i belongs to the following digit set:

$$p_i \in \{-\rho, \ldots, \bar{1}, 0, 1, \ldots, \rho\} \tag{3.46}$$

note that ρ and ρ'' may be different.

The residue digits pass through a RESIDUE Multiplication Unit. The same algorithm (MULT) operates on them, that is, they are multiplied in an on-line mode. The product of the residues will be designated by R' and is defined as:

$$R' = \sum_{i=1}^{m} p'_i r'^{-i}$$

and p'_i belongs to the following set:

$$p'_i \in \{-\rho', \ldots, \bar{1}, 0, 1, \ldots, \rho'\} \tag{3.47}$$

note that even though:

$$x'_i = |x_i|_A \text{ and } y'_i = |y_i|_A$$

but p'_i may not be equal to $|p_i|_A$.

Proof of convergence of the algorithm MULT for the MAIN and RESIDUE operands is the same and is given in Appendix C.

3.2.1.1 The Error Detection Algorithm

The purpose of this section is to develop an algorithm that can detect an error at each step of the on-line multiplication unit. This algorithm will be run after generation of p_i and p'_i and will determine whether these product digits are legal or not.

To derive this algorithm, from Eq. (C.8) in Appendix C, we have:

$$r^{-j}P_j = X_j Y_j - R_{j-1} \tag{3.48}$$

Following a similar procedure as that given in Appendix C, for the RESIDUE Unit we get:

$$r'^{-j}P'_j = X'_j Y'_j - R'_{j-1} \tag{3.49}$$

where X_j, Y_j, X'_j, Y'_j, R_{j-1} and R'_{j-1} are defined below:

$$X_j = \sum_{i=1}^{j} x_i r^{-i} \quad \text{and} \quad Y_j = \sum_{i=1}^{j} y_i r^{-i}$$

$$X'_j = \sum_{i=1}^{j} x'_i r'^{-i} \quad \text{and} \quad Y'_j = \sum_{i=1}^{j} y'_i r'^{-i}$$

$$R_{j-1}=\sum_{i=1}^{j-1} p_i r^{-i} \quad \text{and} \quad R'_{j-1}=\sum_{i=1}^{j-1} p'_i r'^{-i}$$

Taking the residues of X_j, X'_j, Y_j and Y'_j with respect to A we get:

$$|X_j|_A=\left|\sum_{i=1}^{j} x'_i |r|_A^{-i}\right|_A$$

$$|X'_j|_A=\left|\sum_{i=1}^{j} x'_i |r'|_A^{-i}\right|_A$$

if we assume $|r|_A=|r'|_A$ then:

$$|X_j|_A=|X'_j|_A$$

and similarly:

$$|Y_j|_A=|Y'_j|_A$$

Rearranging Equations (3.48) and (3.49) we obtain:

$$\begin{cases} r^{-j}P_j+R_{j-1}=X_jY_j \\ r'^{-j}P'_j+R'_{j-1}=X'_jY'_j \end{cases}$$

Taking the residues of both sides with respect to A:

$$\left|\,|r^{-j}|P_j|_A+|R_{j-1}|_A\right|_A=\left|\,|X_j|_A{}^*|Y_j|_A\right|_A$$

$$=\left|\,|X'_j|_A{}^*|Y'_j|_A\right|_A$$

and

$$\left|\,|r'^{-j}|P'_j|_A+|R'_{j-1}|_A\right|_A=\left|\,|X'_j|_A{}^*|Y'_j|_A\right|_A$$

Therefore:

$$\left|\left|r^{-j}\left|P_j\right|_A+\left|R_{j-1}\right|_A\right|_A=\left|\left|r'^{-j}\left|P'_j\right|_A+\left|R'_{j-1}\right|_A\right|_A \quad (3.50)$$

This is the relation that we check at each step of the multiplication algorithm for correctness of the MAIN and RESIDUE Units. To simplify the checking process we assume:

$$|r|_A=|r'|_A=1 \quad (3.51)$$

Therefore, (3.50) reduces to:

$$\left|\left|P_j\right|_A+\left|R_{j-1}\right|_A\right|_A=\left|\left|P'_j\right|_A+\left|R'_{j-1}\right|_A\right|_A \quad (3.52)$$

The algorithm of the next page is run by the CHECK Unit. The inputs of this unit are (P_j, p_{j-1}) from the MAIN and (P'_j, p'_{j-1}) from the RESIDUE Units.

Algorithm "DETECT MULT"

Step 1 [Initialization]:

$R_{-1}=0$, $p_0=0$

$R'_{-1}=0$, $p'_0=0$

For j=1,2,.....,m+1 Do:

Step 2 [Input Digits]:

$R_{j-1}=|R_{j-2}+p_{j-1}|_A$

$R'_{j-1}=|R'_{j-2}+p'_{j-1}|_A$

P_j and P'_j

Step 3 [Check for Error]:

$Z_j=\left|\,|P_j|_A+R_{j-1}\,\right|_A$

$Z'_j=\left|\,|P'_j|_A+R'_{j-1}\,\right|_A$

If $(Z_j \neq Z'_j)$ E=1, GOTO ERROR SUB

Step 4 [End Do]

3.2.1.2 Radix of the RESIDUE Unit

The bounds on the radix of the RESIDUE Multiplication Unit (r') is similar to that derived for the RESIDUE Division Unit. For the corresponding formulas see Section (3.1.1.3).

3.2.1.3 Bounds On Operands

According to (3.44) and (3.45) we have:

$$M \geq X,\ Y \geq -M$$

$$M' \geq X',\ Y' \geq 0$$

Since the operands of the MAIN Unit are assumed to be redundant, from Eq. (C.22) in Appendix C we have:

$$M \leq \frac{2k-1}{4k''} \tag{3.53}$$

The case of non-redundant operand multiplication has not been addressed in Appendix C of this thesis. But, for this case with a similar derivation the following equation has been obtained:

$$M' \leq k' - \frac{1}{2} \tag{3.54}$$

Note:

After adjusting the operands of the MAIN Unit, if X' and Y' are still out of bounds, multiples of the check

constant (A) can be added to or subtracted from the digits of X' and Y' without changing the results.

3.2.1.4 An Example of The Error-Detection Process

The following is a numerical example of the error detection process when residue-coded operands are used:

Assume:

$$\begin{cases} r=10,\ \rho=5,\ \rho''=9 \\ r'=4,\ \rho'=2 \\ A=3 \end{cases}$$

From (3.42), (3.46), (3.43) and (3.47) we get:

$$\begin{cases} x_i,\ y_i \leftarrow \{\bar{9},\bar{8},\ldots,\bar{1},0,1,\ldots,8,9\} \\ p_i \leftarrow \{\bar{5},\bar{4},\ldots,\bar{1},0,1,\ldots,4,5\} \\ x'_i,\ y'_i \leftarrow \{0,1,2\} \\ p'_i \leftarrow \{\bar{2},\bar{1},0,1,2\} \end{cases}$$

Therefore:

$$k=\frac{5}{9},\ k''=1 \text{ and } k'=\frac{2}{3}$$

From (3.53) and (3.54) we get:

$$M \leq 0.028 \text{ and } M' \leq 0.167$$

Assume M=0.01 and M'=0.167. The operands and their residues are assumed to be:

$$\begin{cases} (X,X')=.(0,0)(0,0)(9,0)(\overline{4},2)(6,0)(\overline{8},1)(9,0) \\ (Y,Y')=.(0,0)(0,0)(\overline{9},0)(7,1)(\overline{2},1)(9,0)(\overline{6},0) \end{cases}$$

Using Eq. (C.21) in Appendix C, the following P-P plot for the selection of the product digits of the MAIN Unit will be obtained.

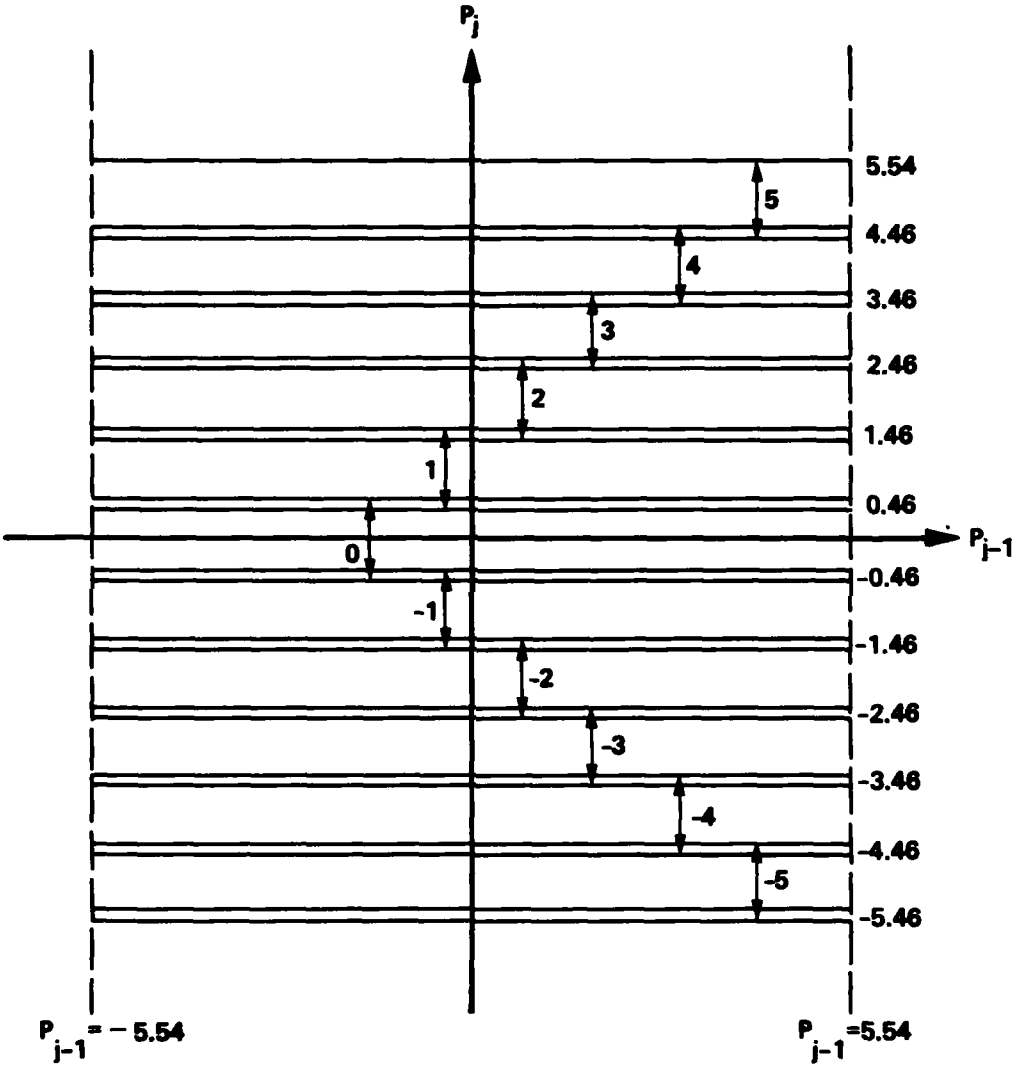


Figure 3.6 – P–P Plot for the MAIN Unit

Using the selection diagram shown in Figure (3.6) and the Algorithm "MULT" given in Appendix C, Table (3.5) is obtained for the operation of the MAIN Unit.

j	X_j	Y_j	P_j	p_j
1	.0	.0	.0	0
2	.0	.0	.0	0
3	.009	.00$\bar{9}$	-.081	0
4	.009$\bar{4}$	.00$\bar{9}$7	-.7138	$\bar{1}$
5	.009$\bar{4}$6	.00$\bar{9}$7$\bar{2}$	2.79488	3
6	.009$\bar{4}$6$\bar{8}$	.00$\bar{9}$7$\bar{2}$9	-1.906772	$\bar{2}$
7	.009$\bar{4}$6$\bar{8}$9	.00$\bar{9}$7$\bar{2}$9$\bar{6}$	.8055636	1

Table (3.5)- Results Obtained by the MAIN Unit (EX. 3.2.1.4)

Also:

$$P_8=P_7-p_7=-0.1944364$$

$$R=0.000\bar{1}3\bar{2}1$$

Therefore:

$$X*Y = 0.000\bar{1}3\bar{2}1\bar{2}1\overline{44}4\bar{4}$$

Figure (3.7) shows the selection digram of the RESIDUE Unit. Table (3.6) shows the results obtained by the RESIDUE Unit.

From this table we get:

$$P'_8=P'_7-p'_7=(0.\overline{1}2\overline{1}1000)_4$$

Therefore:

$$R'=.0000001$$

and

$$X'*Y'=0.0000001\overline{1}2\overline{1}1000$$

Table (3.7) is obtained by using the Algorithm "DETECT MULT" and summarizes the operation of the CHECK Unit for this example.

Since $Z_j=Z'_j$ for j=1,2,...,m, m+1 then all the operations have been correct or an undetectable error has occurred.

Now assume at step 8 of the RESIDUE MULT Algorithm an error changes the sign of the eighth partial product $[P'_8=(0.\overline{1}2\overline{1}1000)_4]$. The incorrect partial product (P'^{*}_8) will be:

$$P'^{*}_8=(0.12\overline{11}000)_4$$

Continuing "DETECT MULT" from step 8 we get:

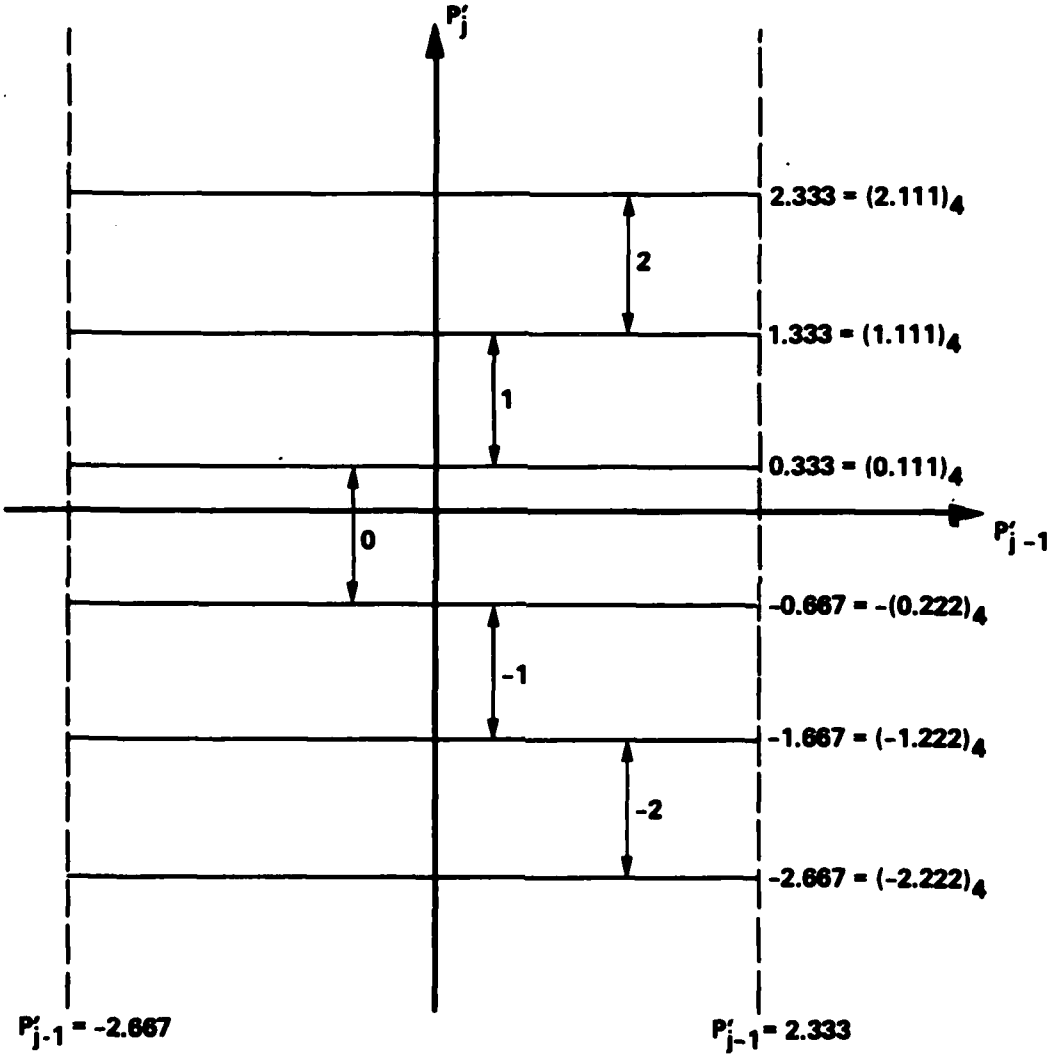


Figure 3.7 – P-P Plot for the RESIDUE Unit

j	X'_j	Y'_j	P'_j	p'_j
1	.0	.0	.0	0
2	.0	.0	.0	0
3	.0	.0	.0	0
4	.0002	.0001	.0002	0
5	.00020	.00011	.0022	0
6	.000201	.000110	.02211	0
7	.0002010	.0001100	.2211	1

Table (3.6)- Results Obtained by the RESIDUE Unit

j=8

$$R_7=|0+1|_A=1$$

$$R'_7=|0+1|_A=1$$

$$Z_8=|2+1|_A=0$$

$$Z'^{*}_8=|1+1|_A=2$$

Since $Z_8 \neq Z'^{*}_8$ this error will be detected by the CHECK Unit.

If no error is detected by the CHECK Unit, the current product digit (p_j) is delivered to the next on-line unit along with its residue modulo A. These two constitute one of

j	R_{j-1}	R'_{j-1}	Z_j	Z'_j	$Z_j=Z'_j$
1	0	0	0	0	check
2	0	0	0	0	check
3	0	0	0	0	check
4	0	0	2	2	check
5	2	0	1	1	check
6	2	0	0	0	check
7	0	0	0	0	check
8	1	1	0	0	check

Table (3.7)- Results Obtained by the CHECK Unit

the operands of the following on-line unit.

3.2.2 AN Coded Operands

the purpose of this section is to present AN Codes in the process of on-line multiplication. As was mentioned earlier, the operands and the results are shown with m digits in a radix-r redundant number system. They are denoted by X,Y and R for the multiplicand, multiplier and the product respectively. The encoded operands are obtained by just multiplying each digit of the operands by the check modulus (A). Table of the next page shows the correspondence between the two sets of operands and the results. Note that since each digit of the multiplicand and the multiplier is a multiple of A, then the product digits will be multiples of A^2 and not A. Therefore, at the end of each step each product digit should be dividend by A to get the correctly encoded product. If we assume that A is a low-cost modulus, this operation will be trivial [AVI 73].

$$X=\sum_{i=1}^{m} x_i r^{-i} \quad \text{and} \quad X'=\sum_{i=1}^{m} x'_i r^{-i}$$

$$Y=\sum_{i=1}^{m} y_i r^{-i} \quad \text{and} \quad Y'=\sum_{i=1}^{m} y'_i r^{-i}$$

$$R=\sum_{i=1}^{m} p_i r^{-i} \quad \text{and} \quad R'=\sum_{i=1}^{m} p'_i r^{-i}$$

$$\begin{cases} x_i \in \{-\rho',\ldots,\bar{1},0,1,\ldots,\rho'\} \text{ and} \\ x'_i \in \{-A\rho',\ldots,\bar{A},0,A,\ldots,A\rho'\} \end{cases} \tag{3.55}$$

$$\begin{cases} y_i \in \{-\rho',\ldots,\bar{1},0,1,\ldots,\rho'\} \text{ and} \\ y'_i \in \{-A\rho',\ldots,\bar{A},0,A,\ldots,A\rho\} \end{cases} \tag{3.56}$$

$$\begin{cases} p_i \in \{-\rho,\ldots,\bar{1},0,1,\ldots,\rho\} \text{ and} \\ p'_i \in \{-A^2\rho,\ldots,-A^2,0,A^2,\ldots,A^2\rho\} \end{cases} \tag{3.57}$$

$$-M \leq X,Y \leq M \quad \text{and} \quad -AM \leq X',Y' \leq AM \tag{3.58}$$

and the relation between the corresponding digits of these two sets of operands and the results are:

$$\begin{cases} x'_i = Ax_i \\ y'_i = Ay_i \qquad i=1,\ldots,m \\ p'_i = A^2 p_i \end{cases} \tag{3.59}$$

The algorithm that operates on encoded operands is the algorithm "MULT" shown in Appendix C. The proof of the convergence of the algorithm to the correct value of the product is similar to that shown for the algorithm "MULT". Similar to Eq. (C.10) we get:

$$R'=X'Y'-r^{-m}(P'_m-p'_m) \tag{3.60}$$

By devising a product digit selection procedure SELECT, in step 4 of the algorithm "MULT" such that:

$$|P'_m-p'_m|\leq A^2k \tag{3.61}$$

R'=X'*Y' can be computed to m digit precision. The least significant half of the product is available as the redundant output of the adder after iteration m+1, i.e.,

$$P'_{m+1}=P'_m-p'_m \tag{3.62}$$

3.2.2.1 Selection of The Product Digits

Selection of the correct product digit is of great importance in the design of the AN-coded units. Looking back into Eq. (3.57) we deduce that the correct product digit is always a multiple of the square of the check modulus. Derivation of the bounds on the encoded partial product follows similar path as that explained in Appendix C. These bounds are:

$$A^2(rk-2Mk') \geq P'_j \geq A^2(-rk+2Mk') \qquad (3.62)$$

selection equations are also represented by a P'-P' plot. It consists of a family of curves which are linear function of P'_{j-1} with p'_j as parameter ranging from $-A^2\rho$ to $+A^2\rho$ in step of A^2. The area between maximum P'_j and minimum P'_j will be denoted by the $p'_j=i'$ region. A given value of P'_{j-1} and P'_j will correspond to a point in an i' selection region. The product digit p'_j is, therefore, i' and is used in forming the next partial product. The following equation shows these regions:

$$i'+A^2k-2Mk'A^2 \geq (P'_j)^{p'_j=i'} \geq i'-A^2k+2Mk'A^2 \qquad (3.63)$$

when j=m in (3.63):

$$A^2k-2Mk'A^2 \geq P'_m-p'_m \geq -A^2k+2Mk'A^2$$

and since $Mk'A^2>0$ then:

$$A^2k \geq P'_m-p'_m \geq -A^2k$$

or

$$|P'_m-p'_m| \leq A^2k$$

Therefore the relation (3.61) is satisfied by the above selection equations. This proves that R' is indeed the correct product up to m digits.

3.2.2.2 Bounds on Operands

Allowable values of M are obtained by requiring that the upper bound of the $p'_j=i'-A^2$ selection region be always greater than the lower bound of the $p'_j=i'$ region, i.e.

$$U_{i'-A^2} \geq L_{i'}$$

inserting the values from Eq. (3.63) we get:

$$M \leq \frac{2k-1}{4k'} \tag{3.64}$$

This is exactly the same bound we obtained in Appendix C (Eq. C.22). Therefore, applying AN-Codes to the operands does not change the allowable range.

3.2.2.3 An Example of Multiplication With AN-Coded Operands

Assume:

$$r=2,\ k=k'=1 \text{ and } A=3$$

applying Eq. (3.64) we get $M \leq \frac{1}{4}$.

assume $M=\frac{1}{4}$. Equations (3.55) to (3.58) result in:

$$\begin{cases} x'_i,\ y'_i \in \{\bar{3},0,3\} \\ p'_i \in \{\bar{9},0,9\} \\ -\frac{3}{4} \leq X', Y' \leq \frac{3}{4} \end{cases}$$

inserting the given values into Eq. (3.63) we get:

$$i'+\frac{9}{2} \geq P'_j \geq i'-\frac{9}{2} \qquad i' \in \{\bar{9},0,9\}$$

Figure (3.8) depicts this set of equations.

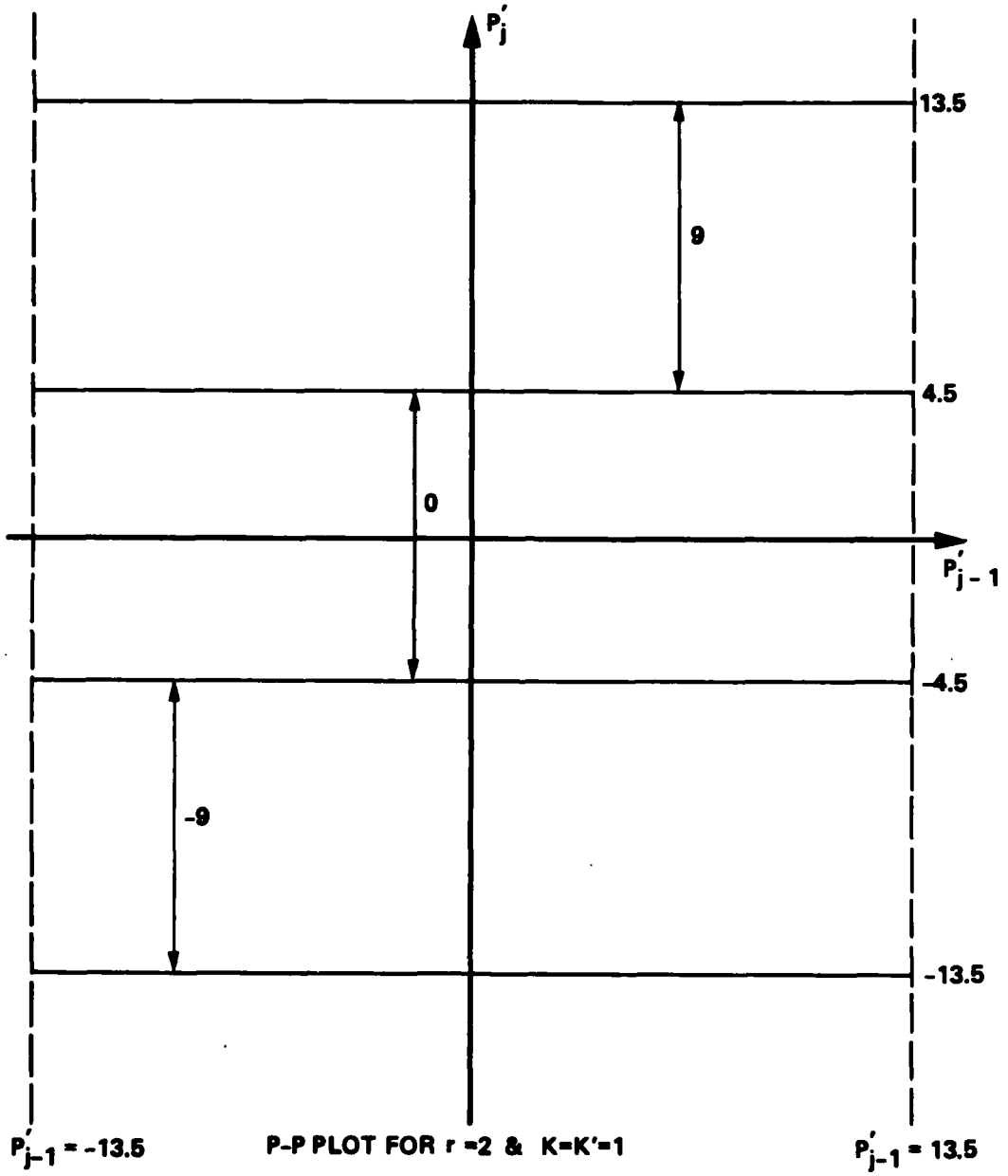


Figure 3.8 – P-P Plot for the AN-Coded Unit

As an example assume that the multiplicand and the multiplier are:

$$\begin{cases} X' = .00\bar{3}03\bar{3}30\bar{3} \\ Y' = .00\bar{3}0303\bar{3}30 \end{cases}$$

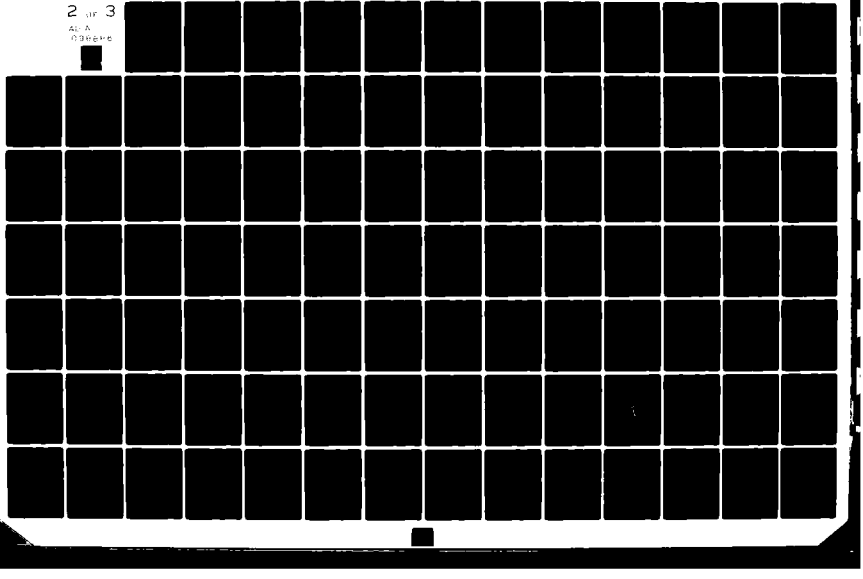
Following the steps of the algorithm "MULT" in Appendix C, Table (3.8) will result.

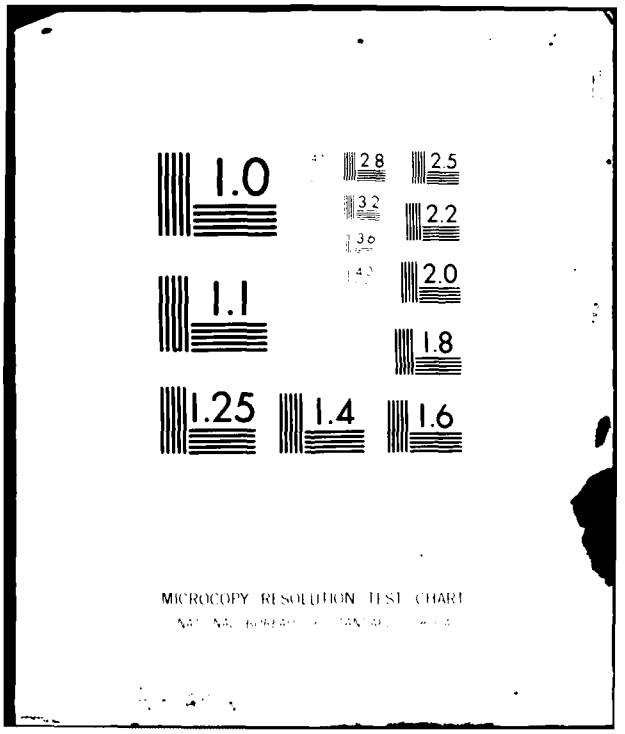
j	X'_j	Y'_j	P'_j	p'_j
1	.0	.0	.0	0
2	.0	.0	.0	0
3	.003	$.00\bar{3}$	$.00\bar{9}$	0
4	$.003\bar{3}$	$.00\bar{3}0$	$.00\bar{9}$	0
5	$.00\bar{3}30$	$.00\bar{3}03$	$.00\bar{99}$	0
6	$.00\bar{3}303$	$.00\bar{3}030$	$.0\bar{9}\bar{9}\bar{9}\bar{9}$	0
7	$.00\bar{3}03\bar{3}$	$.00\bar{3}0303$	$.\bar{9}\bar{9}00\bar{9}\bar{9}$	$\bar{9}$
8	$.00\bar{3}0\bar{3}\bar{3}3$	$.00\bar{3}03033$	.09009099	0
9	$.00\bar{3}03\bar{3}330$	$.00\bar{3}0303\bar{3}3$	.90909009	9
10	$.00\bar{3}03\bar{3}30\bar{3}$	$.00\bar{3}0303\bar{3}30$	$.\bar{9}00\bar{9}0\bar{9}\bar{9}\bar{9}\bar{9}$	$\bar{9}$

Table (3.8)- An Example of AN-Coded Multiplication

From this table we get:

AD-A098 886 CALIFORNIA UNIV LOS ANGELES DEPT OF COMPUTER SCIENCE F/G 9/2
ERROR-CODED ALGORITHMS FOR ON-LINE ARITHMETIC.(U)
FEB 81 A GORJI-SINAKI N00014-79-C-0866
UNCLASSIFIED UCLA-ENG-8197 NL




1.0
2.8
2.5
3.2
2.2
3.6
2.0
1.1
1.8
1.25
1.4
1.6
MICROCOPY RESOLUTION TEST CHART

$$P'_{11}=P'_{10}-p'_{10}=(.0990900090)_2$$

and therefore:

$$X'*Y'=(.000000\bar{9}09\bar{9}0990900090)_2$$

The necessary condition for correctness of the operation is satisfied because all the digits of the product and partial product are multiples of the check modulus (A=3).

3.3 Error-Coded On-line Addition

3.3.1 Residue-Coded Operands

Assume that the summands A and B are represented by m digits in a redundant number representation system (Eqs. D.1 and D.2). The residue encoded summands are (A, A') and (B, B') such that:

$$(A,\ A')=.(a_1,a'_1)(a_2,a'_2)\ldots(a_m,a'_m)$$

$$(B,\ B')=.(b_1,b'_1)(b_2,b'_2)\ldots(b_m,b'_m)$$

The relation between A and A' (B and B') is:

$$a'_i=|a_i|_A$$

$$b'_i=|b_i|_A$$

a_i and b_i are assumed to belong to the following digit set:

$$a_i,b_i \in \{-\rho'',\ldots,\bar{1},0,1,\ldots,\rho''\} \quad (r-1)\geq\rho''\geq r/2 \qquad (3.65)$$

The following relation is obtained directly from the definition of residue function.

$$a'_i,b'_i \in \{0,1,\ldots,(A-1)\} \qquad (3.66)$$

The sum R is shown by m+1 digits also in a radix-r redundant number system (Eq. D.3). the residue encoded sum is:

$$(R,|R|_A)=(s_0,|s_0|_A)\ .\ (s_1,|s_1|_A)....(s_m,|s_m|_A)$$

s_i is assumed to belong to the following set:

$$s_i \leftarrow \{-\rho,...,\bar{1},0,1,...,\rho\} \quad (r-1)\geq \rho \geq r/2 \qquad (3.67)$$

A' and B' in going through the RESIDUE Unit generate a residue sum which is represented by R' such that:

$$R'=\sum_{i=0}^{m} s'_i r'^{-i}$$

s'_i is assumed to belong to the following digit set:

$$s'_i \leftarrow \{-\rho',...,\bar{1},0,1,...,\rho'\} \quad (r'-1)\geq \rho' \geq r'/2 \qquad (3.68)$$

The algorithm run by the MAIN and RESIDUE Units is the algorithm "ADD" presented in Appendix D. Proof of the convergence of this algorithm to the correct value of the sum is given in the same appendix.

3.3.1.1 The Error Detection Algorithm

In this section the algorithm which should be run by the CHECK Unit will be derived. CHECK Unit starts the operation after generation of s_i and s'_i by the MAIN and RESIDUE Units, respectively. It examines the necessary condition for fault free operation of the MAIN and RESIDUE Units. Unless this necessary condition is satisfied, the CHECK Unit stops the operation and sets an error flag.

To derive such an algorithm, from equation (D.9) of Appendix D we get:

$$\sum_{i=1}^{j}(a_i+b_i)r^{-i} = \sum_{i=0}^{j-2}s_i r^{-i} + r^{-j+1}p_j \qquad (3.69)$$

For the RESIDUE Unit we get a similar expression:

$$\sum_{i=1}^{j}(a'_i+b'_i)r'^{-i} = \sum_{i=0}^{j-2}s'_i r'^{-i} + r'^{-j+1}p'_j \qquad (3.70)$$

Taking the residues of (3.69) and (3.70) we obtain (3.71) and (3.72).

$$\left|\sum_{i=1}^{j}(a'_i+b'_i)r^{-i}\right|_A = \left|\left|\sum_{i=0}^{j-2}s_i r^{-i}\right|_A + r^{-j+1}|p_j|_A\right|_A \qquad (3.71)$$

and

$$\left|\sum_{i=1}^{j}(a'_i+b'_i)r'^{-i}\right|_A = \left|\left|\sum_{i=0}^{j-2}s'_i r'^{-i}\right|_A + r'^{-j+1}|p'_j|_A\right|_A \qquad (3.72)$$

Assuming $|r|_A=|r'|_A$, from (3.71) and (3.72) we get:

$$\left|\left|\sum_{i=0}^{j-2}s_i r^{-i}\right|_A + r^{-j+1}|p_j|_A\right|_A =$$

$$\left|\left|\sum_{i=0}^{j-2}s'_i r^{-i}\right|_A + r^{-j+1}|p'_j|_A\right|_A \qquad (3.73)$$

This is the relation that CHECK Unit verifies at the end of each step. To further simplify the detection process we assume $|r|_A=|r'|_A=1$. As a result, (3.73) reduces to:

$$\left|\left|\sum_{i=0}^{j-2} s_i\right|_A + |P_j|_A\right|_A = \left|\left|\sum_{i=0}^{j-2} s'_i\right|_A + |P'_j|_A\right|_A \qquad (3.74)$$

Defining the following dummy variables:

$$X_{j-2} = \left|\sum_{i=0}^{j-2} s_i\right|_A$$

$$X'_{j-2} = \left|\sum_{i=0}^{j-2} s'_i\right|_A$$

Eq. (3.74) becomes:

$$\left|X_{j-2}+|P_j|_A\right|_A = \left|X'_{j-2}+|P'_j|_A\right|_A \qquad (3.75)$$

The algorithm "DETECT ADD" of the next page is run by the CHECK Unit and verifies Eq. (3.75).

Algorithm "DETECT ADD"

Step 1 [Initialization]:

$X_{-2}=0,\ X'_{-2}=0$

For j=1,2,....,m+2 Do:

Step 2 [Input Digits]:

$X_{j-2}=|X_{j-3}+s_{j-2}|_A$

$X'_{j-2}=|X'_{j-3}+s'_{j-2}|_A$

$|P_j|_A$ and $|P'_j|_A$

Step 3 [Check for Error]:

$Z_{j-2}=|X_{j-2}+|P_j|_A|_A$

$Z'_{j-2}=|X'_{j-2}+|P'_j|_A|_A$

If $(Z_{j-2} \neq Z'_{j-2})$

THEN E=1, GOTO ERROR SUB

Step 4 [End Do]

3.3.1.2 An Example of The Error Detection Process

In what follows a numerical example of the residue-coded addition will be given.

Assume:

$$\begin{cases} r=10,\ \rho=\rho''=9 \\ r'=4,\ \rho'=3 \\ A=3 \end{cases}$$

Therefore:

$$k=k'=k''=1$$

From Eqs. (3.65), (3.66), (3.67) and (3.68) we have:

$$a_i, b_i, s_i \leftarrow \{\overline{9}, \overline{8}, \ldots, \overline{1}, 0, 1, \ldots, 8, 9\}$$

$$a'_i, b'_i \leftarrow \{0, 1, 2\}$$

$$s'_i \leftarrow \{\overline{3}, \overline{2}, \overline{1}, 0, 1, 2, 3\}$$

The encoded summands (A,A') and (B,B') are assumed to be:

$$(A,A') = .(9,0)(2,2)(4,1)(7,1)(0,0)(5,2)$$

$$(B,B') = .(8,2)(4,1)(3,0)(5,2)(6,0)(7,1)$$

Following the algorithm "ADD" in Appendix D, Table (3.9) will be obtained. This table summarizes the results obtained by the MAIN Addition Unit during various steps of the ADD algorithm. According to this table:

j	P_j	s_{j-1}	R_j
1	1.7	2	2
2	-2.4	$\bar{2}$	1.8
3	-3.3	$\bar{3}$	1.77
4	-1.8	$\bar{2}$	1.768
5	2.6	3	1.7683
6	-2.8	$\bar{3}$	1.76827
7	2.0	2	1.768272

Table (3.9)- Results Obtained by the MAIN Unit (EX. 3.3.1.2)

$R=R_7=1.768272$

Table (3.10) summarizes the results obtained by the RESIDUE Addition Unit. From this table we get:

$$R'=R'_7=(0.3\bar{1}2\bar{1}1\bar{1})_4=(.231303)_4$$

The information needed by the CHECK Unit are: s_{j-2} , s'_{j-2} , $|P_j|_A$ and $|P'_j|_A$. Table (3.11) is obtained by this unit using the algorithm "DETECT ADD" and summarizes the operation of the CHECK Unit for this example.

This table indicates that the necessary condition for correctness of the operation is satisfied for every step of

j	P'_j	s'_{j-1}	R'_j
1	0.2	0	.0
2	0.23	3	.3
3	-0.3	$\overline{1}$	.23
4	1.3	2	.232
5	-1.0	$\overline{1}$	.2313
6	0.3	1	.23131
7	-1.0	$\overline{1}$	.231303

Table (3.10)- Results Obtained by the RESIDUE Unit

the ADD algorithm ($Z_{j-2}=Z'_{j-2}$ for j=1,..,8).

In order to demonstrate the error detection capability of the proposed scheme, assume due to an error in the multi-input adder of the MAIN Unit, the sign bit of P_6 has been inverted. Such that:

$$P_6=-2.8 \Rightarrow |P_6|_A=2$$

$$P_6^*=2.8 \Rightarrow |P_6^*|_A=1$$

Following the "DETECT ADD" algorithm from step j=6 we get:

j=6

j	X_{j-1}	X'_{j-1}	Z_{j-1}	Z'_{j-1}	$Z_{j-1}=Z'_{j-1}$
1	0	0	2	2	check
2	2	0	2	2	check
3	0	0	0	0	check
4	0	2	0	0	check
5	1	1	0	0	check
6	1	0	0	0	check
7	1	1	0	0	check
8	0	0	0	0	check

Table (3.11)- Results Obtained by The CHECK Unit

$X_4=|1+3|_3=1$

$X'_4=|1+\bar{1}|_3=0$

$Z^*_4=|1+1|_3=2$

$Z'_4=|0+0|_3=0$

Since $Z^*_4 \neq Z'_4$, this error will be detected by the CHECK Unit.

3.3.2 AN Coded Operands

In this section the imposition of AN codes on on-line addition will be considered. On-line subtraction can be replaced by addition by just flipping the signs of the subtrahend digits. As before, the encoded summands (A' and B') are obtained by multiplying each of their digits by the check modulus.

$$A' = \sum_{i=1}^{m} (Aa_i) r^{-i} = \sum_{i=1}^{m} a'_i r^{-i}$$

$$B' = \sum_{i=1}^{m} (Ab_i) r^{-i} = \sum_{i=1}^{m} b'_i r^{-i}$$

$$R' = \sum_{i=0}^{m} (As_i) r^{-i} = \sum_{i=0}^{m} s'_i r^{-i}$$

a'_i , b'_i and s'_i belong to the following digit sets:

$$a'_i \in \{-A\rho', \ldots, \bar{A}, 0, A, \ldots, A\rho'\} \tag{3.76}$$

$$b'_i \in \{-A\rho'', \ldots, \bar{A}, 0, A, \ldots, A\rho''\} \tag{3.77}$$

$$s'_i \in \{-A\rho, \ldots, \bar{A}, 0, A, \ldots, A\rho\} \tag{3.78}$$

It is clear that:

$$Ak' > A' > -Ak'$$

$$Ak'' > B' > -Ak''$$

$$A(k'+k'') > R' > -A(k'+k'')$$

Algorithm "ADD", shown in Appendix D, is imposed on the encoded operands A' and B'. With a similar derivation the following relation will be obtaned:

$$R'=(A'+B') - r^{-m} (P_{m+1}-s'_m) \tag{3.79}$$

The sum digit selection procedure in step 4 of the algorithm "ADD" should be such that the following relation is satisfied:

$$|P'_{m+i}-s'_m|<Ak' \tag{3.80}$$

Only in that case R' represents the sum of A' and B' to m digits of precision.

3.3.2.1 Selection of The Sum Digits

Sum digits should be selected in such a way that they are always multiples of the check modulus (A). Using the basic recursion formula (Eq. D.8 in Appendix D) and following a similar procedure given in that appendix, the bounds on the correct partial sum will be obtained.

$$rkA - \frac{a'_j+b'_j}{r(r-1)} \geq P_j \geq -rkA - \frac{a'_j+b'_j}{r(r-1)} \tag{3.81}$$

The selection region i', is a region in which $s'_{j-1}=i'$ is a correct sum digit. This area is represented by the following inequality.

$$i'+kA - \frac{a'_j+b'_j}{r(r-1)} \geq (P'_j)^{s'_{j-1}=i'} \geq i'-kA - \frac{a'_j+b'_j}{r(r-1)} \qquad (3.82)$$

In deriving this relation we have followed a similar path shown in Appendix D for deriving Eq. (D.23).

In order to have overlaps between regions $s'_{j-1}=i'$ and $s'_{j-1}=i'+A$ the following inequality should hold:

$$U_{i'} \geq L_{i'+A} \quad \text{for all values of } i'$$

The only requirement to satisfy this relation is $k \geq \frac{1}{2}$. Therefore, if the sum (R') is in a redundant number representaion system, there are always overlaps between the adjacent regions even if the summands are not redundant.

3.3.2.2 An Example of Addition With AN-Coded Operands

Assume:

$$r=10,\ k=k'=k''=1,\ m=8 \text{ and } A=7$$

From Equations (3.76), (3.77) and (3.78) we get:

$$a'_i,\ b'_i,\ s'_i \in \{\overline{63},\overline{56},\ldots,\overline{7},0,7,\ldots,56,63\}$$

As a numerical example assume the summands (A', B') are:

$$A'=.(63)(56)(\overline{63})(\overline{42})(35)(14)(7)(63)$$

$$B'=.(63)(63)(\overline{63})(\overline{56})(42)(63)(\overline{14})(7)$$

Applying the "ADD" algorithm on these sets of operands, we obtain Table (3.12). From this table the value of sum

j	P'_j	s'_{j-1}
1	(7).(56)	14
2	(0).($\overline{21}$)	0
3	($\overline{28}$).($\overline{56}$)	$\overline{35}$
4	(0).(42)	7
5	($\overline{14}$).($\overline{63}$)	$\overline{21}$
6	(14).(7)	14
7	(0).(63)	7
8	(0).(0)	0
9	(0).(0)	0

Table (3.12)- An Example of AN-Coded Addition

is:

$$R'=(14).(0)(\overline{35})(7)(\overline{21})(14)(7)(0)(0)$$

the necessary condition for the correctness of the operation is satisfied because all the digits of the sum and partial sum are multiples of the check constant (A=7).

CHAPTER 4

IMPLEMENTATION CONSIDERATIONS

In this chapter a hardware organization of the proposed error-coded on-line units will be presented. In Chapter 3 we proposed two methods for detection and/or correction of errors in an on-line unit. These methods were: 1) use of a residue unit along with the residue coded operands, 2) use of AN coded operands along with one (MAIN) unit and the corresponding CHECK unit. It is obvious that in the first case a CHECK unit is also needed to compare the results of the MAIN and RESIDUE processors.

The operation of each of these units has already been explained in Chapter 3 (see the corresponding block diagrams). It is the purpose of the current chapter to consider the hardware realization of each of these units. At the end, using this realization an estimate of the gate and memory requirements of the error-coded on-line unit will be given. In order to do this, we start with the operation of a residue-coded divide unit. The extension of this work to other basic operations (addition/subtraction and multiplication) is straight forward and will not be considered in this thesis.

4.1 Design of The Error-Coded Division Unit

As we mentioned in previous chapters, a residue-coded on-line division unit consists of the following elements:

1. A Radix r ($=2^k$) MAIN Unit
2. A Radix r' ($=2^{k'}$) RESIDUE Unit
3. A CHECK Unit

In what follows the hardware design of each of these components will be considered and an estimate of the cost of each unit will be given.

4.1.1 Design of The Residue-Coded MAIN Unit

The design of the MAIN Unit when no error-detection scheme is used has been given in the Appendix A. In this section we modify this design to make the same unit suitable for the case when residue coded operands are used.

From the algorithm "DETECT DIVIDE" in Chapter 3 it is clear that the residue of the partial remainders P_j and P'_j are needed by the CHECK Unit at every step of the on-line division algorithm. These residues should be obtained by Processing Elements inside the MAIN and the RESIDUE Units in such a way that the modularity of the units is preserved. Having this in mind, the following scheme for determination of $|P_j|_A$ and $|P'_j|_A$ is proposed.

Calculation of The Residues of Partial Remainders

From the description of the hardware design of the Processing Elements (PE's), we know that partial remainders are represented by m digits. Each PE contains one digit (radix 2^k) of the interim partial remainder ($w_i^{(j)}$) and the corresponding transfer function ($T_i^{(j)}$) such that:

$$P_j = \sum_{i=1}^{m} p_i^{(j)} r^{-i} = \sum_{i=1}^{m} \left[w_i^{(j)} + T_i^{(j)} \right] r^{-i} \tag{4.1}$$

and similarly:

$$P'_j = \sum_{i=1}^{m} p'^{(j)}_i r'^{-i} = \sum_{i=1}^{m} \left[w'^{(j)}_i + T'^{(j)}_i \right] r'^{-i} \tag{4.2}$$

Therefore:

$$|P_j|_A = \left| \sum_{i=1}^{m} |p_i^{(j)}|_A * |r^{-i}|_A \right|_A$$

$$= \left| \sum_{i=1}^{m} \left| |w_i^{(j)}|_A + |T_i^{(j)}|_A \right|_A * |r^{-i}|_A \right|_A \tag{4.3}$$

Assuming $|r|_A = |r'|_A = 1$ we get:

$$|P_j|_A = \left| \sum_{i=1}^{m} \left| |w_i^{(j)}|_A + |T_i^{(j)}|_A \right|_A \right|_A$$

$$= \left| \sum_{i=1}^{m} R_i^{(j)} \right|_A \tag{4.4}$$

where $R_i^{(j)}$ is defined as:

$$R_i^{(j)} = \left| |w_i^{(j)}|_A + |T_i^{(j)}|_A \right|_A \tag{4.5}$$

Using Eq. (A.9) in Appendix-A we get:

$$R_i^{(j)} = \left| |w_i^{(j)}|_A + |t_i^{A(j)}|_A + |t_i^{P_1(j)}|_A + |t_i^{P_2(j)}|_A \right|_A \qquad (4.6)$$

$R_i^{(j)}$'s are computed by Processing Elements. These residue digits [α bits each] are sent to the CHECK Unit. This unit adds these residues and finds the residue of the result in order to obtain $|P_j|_A$ according to Eq. (4.4).

PE_i obtains $R_i^{(j)}$ by adding the residues of the values in RW, TA, TP1 and TP2 registers and finding the residue of the result (Eq. 4.6). These values are obtained as described next.

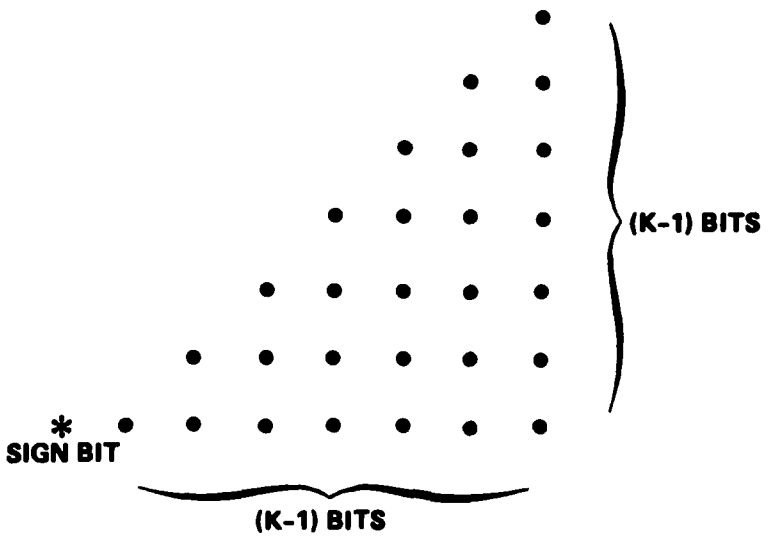
Computation of $w_i^{(j)}$:

Since $w_i^{(j)}$ is a radix-r Sign and Magnitude digit (k+1 bits), its residue is obtained simply by a two-stage ROM device with the capacity of:

$$M_w = \alpha[2^k + 2^{\alpha+1}] \text{ bits} \qquad (4.7)$$

Computation of $t_i^{P_1(j)}$ and $t_i^{P_2(j)}$:

These two transfer functions have a similar form and consist of $\frac{k(k-1)}{2}$ magnitude bits and one sign bit. They are in the following form:



The residues of $t_i^{p_1(j)}$ and $t_i^{p_2(j)}$ are obtained by simply finding the residue of each row and adding them together. This scheme is shown in Figure (4.1) for A=3. Therefore, the total ROM needed for this process is:

$$M_{TP1}=M_{TP2}=[2^{\alpha+1}+2^{\alpha+2}+.....+2^{k-1}]\alpha+[2^{k-1}+2^{k-2}$$

$$+.....+2^{k-\alpha}]\alpha+\alpha*2^{\alpha^2+1}$$

This expression reduces to:

$$M_{TP1}=M_{TP2}=\alpha[2^{k+1}-2^{\alpha+1}-2^{k-\alpha}+2^{\alpha^2+1}] \quad (4.8)$$

The time required for this process is:

$$t_{TP1}=t_{TP2}=3t_M \quad (4.9)$$

where t_M is the read time of a ROM device.

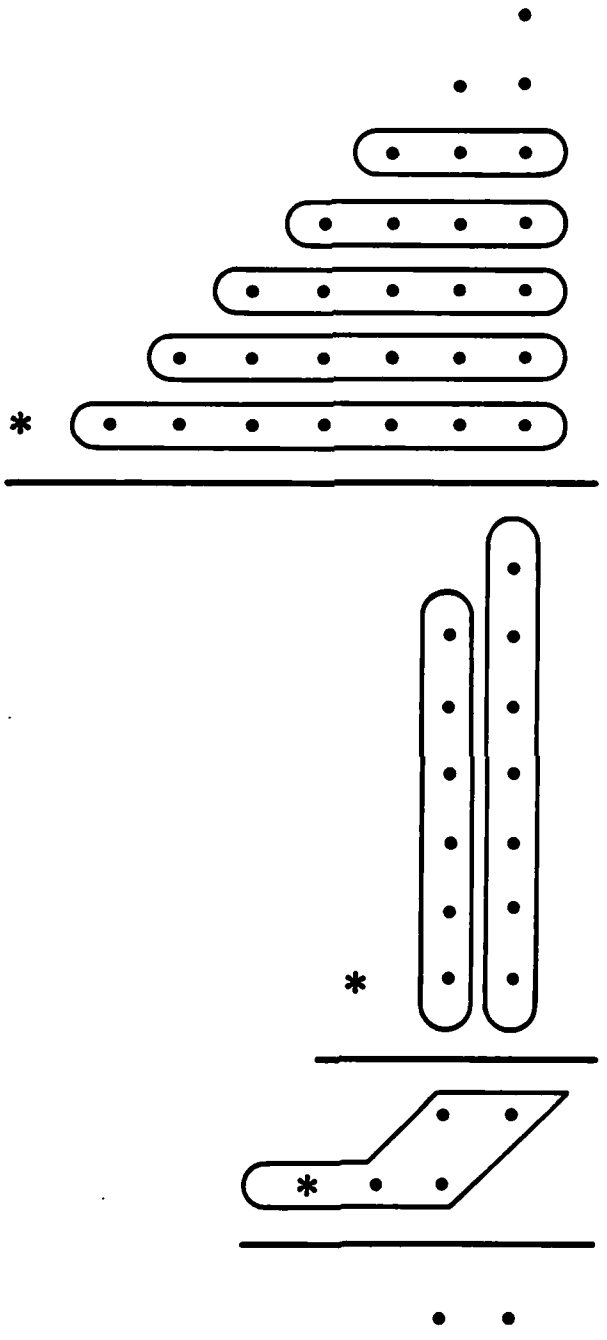


Figure 4.1 – Obtaining Residue of $t_i^{p_1(j)}$

Computation of $t_i^{A(j)}$:

$t_i^{A(j)}$ is the transfer function out of the Multi-Input Adder (MIAD) and is represented by (2k+1) redundant binary digits ($\in\{\bar{1},0,1\}$).

In order to obtain the residue of $t_i^{A(j)}$ with respect to A, (2k+1) digits of it are grouped into groups of 2 digits each. The residue of all groups are obtained simultaneously. the results are grouped again and this process continues until the residue of $t_i^{A(j)}$ is obtained. This scheme is shown in Figure (4.2) for k=8 and A=3.

Number of levels required is:

$$L=\lceil \log_2(2k+1) \rceil \tag{4.10}$$

Therefore, the time required for this process (t_{TA}) is:

$$t_{TA}=Lt_M=\lceil \log_2(2k+1) \rceil t_M \tag{4.11}$$

The memory required (M_{TA}) is:

$$M_{TA}=k2^4*2+\alpha\{\frac{k}{2}2^4+\frac{k}{4}2^{2\alpha}+\frac{k}{8}2^{2\alpha}+.....+2^{2\alpha}\}$$

This function can be approximated by:

$$M_{TA}=32k+8\alpha k+2^{2\alpha}*\alpha*\frac{k-2}{2} \tag{4.12}$$

According to Eq. (4.6) these residues should be added to obtain $R_i^{(j)}$ inside the i-th Processing Element. The following organization is proposed to perform this modular addition.

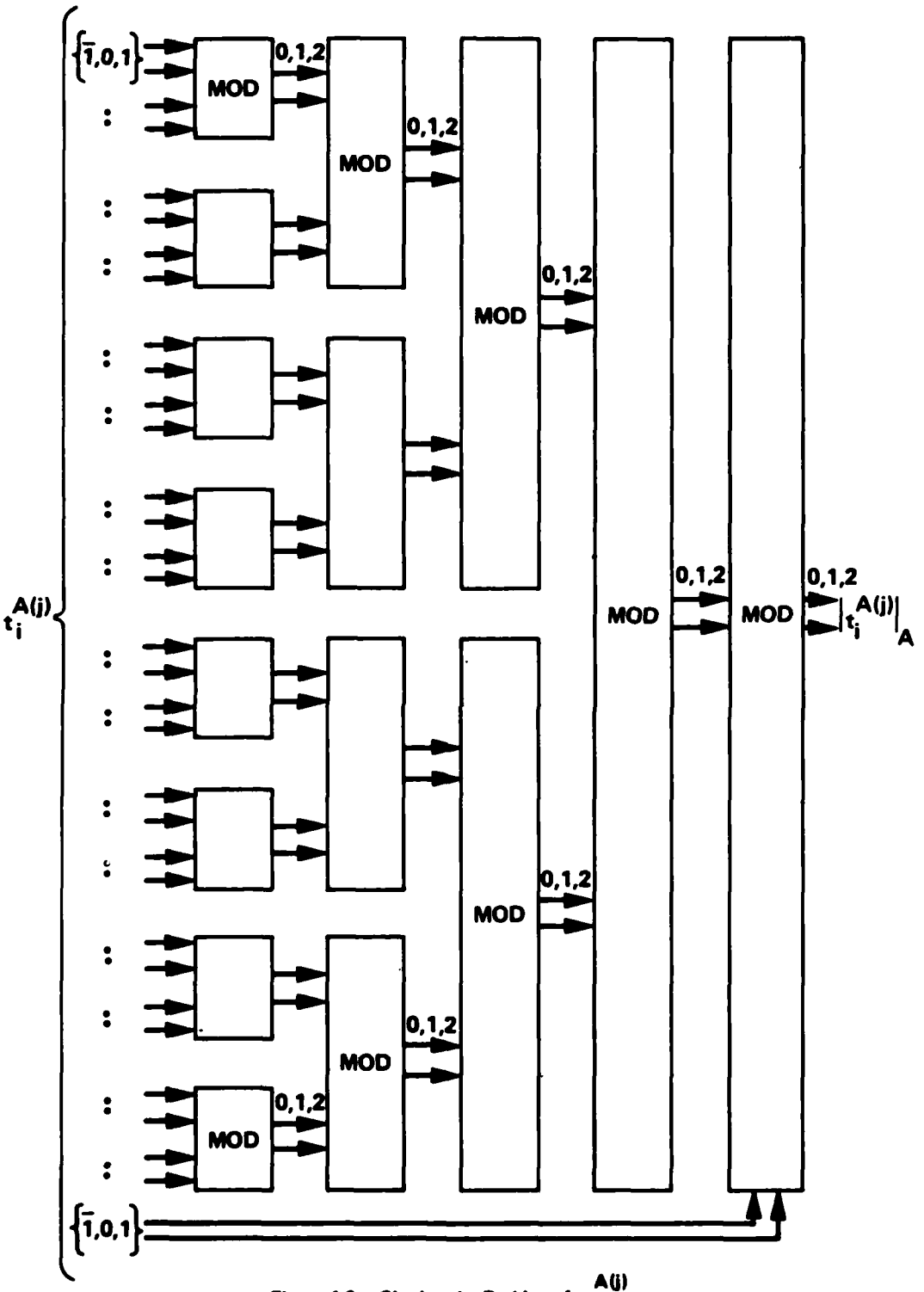
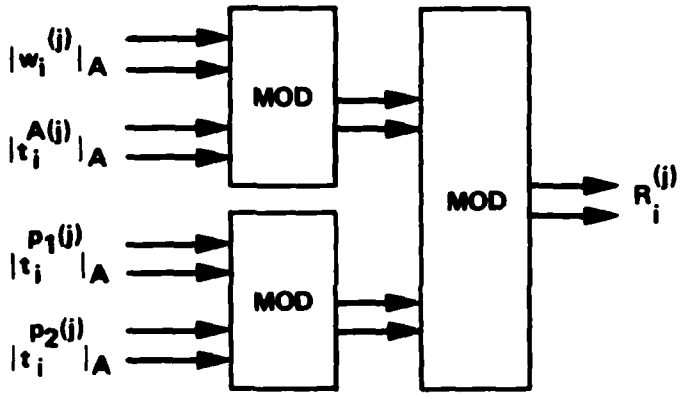


Figure 4.2 – Obtaining the Residue of $t_i^{A(j)}$



The memory required for this process (M_{ADD}) is:

$$M_{ADD}=\alpha\,[3*2^{2\alpha}]=3\alpha\,2^{2\alpha} \qquad (4.13)$$

Therefore, the total ROM required to obtain $R_i^{(j)}$ from $w_i^{(j)}$ and $T_i^{(j)}$ according to Equations (4.7), (4.8), (4.12) and (4.13) is:

$$M_R=\alpha\,2^k[5-2^{-\alpha+1}]+\alpha\,[8k+2^{\alpha^2+2}-2^{\alpha+1}]$$

$$+\alpha\,2^{2\alpha}\,\frac{k+4}{2}+32k \quad \text{(bits)} \qquad \text{for } k>\alpha+1 \qquad (4.14)$$

The time required for this process is:

$$t_r=\left\{\text{MAX}\left(3,\left\lceil\log_2(2k+1)\right\rceil\right)+2\right\}t_M \qquad (4.15)$$

4.1.2 Design of The RESIDUE Unit

Since the RESIDUE and the MAIN Units are similar, from Eq. (A.30) in Appendix-A we get:

$$G_{PE'}=64k'^2+157k'+123 \qquad (4.16)$$

and also the pin requirements of a Processing Element of the RESIDUE Unit is:

$$P_{PE'}=13k'+9$$

The amount of memory required to compute $R'^{(j)}_i$ from $w'^{(j)}_i$ and $T'^{(j)}_i$ similar to Eq. (4.14) is:

$$M_{R'}=\alpha\, 2^{k'}[5-2^{-\alpha+1}] +\alpha\, [8k'+2^{\alpha^2+2}-2^{\alpha+1}]$$

$$+\alpha\, 2^{2\alpha}\,\frac{k'+4}{2} +32k' \quad \text{(bits)} \quad \text{for } k' > \alpha+1 \qquad (4.18)$$

and also the time is similar to Eq. (4.15).

4.1.3 Design of The CHECK Unit

The CHECK Unit receives its inputs from the MAIN and RESIDUE Units. These inputs include:

1. The corresponding digit of the dividend and its residue (n_i and n'_i)
2. The corresponding digit of the divisor and its residue (d_i and d'_i)
3. Output digits of the MAIN and RESIDUE Units (q_i and q'_i)
4. $R^{(j)}_i$ and $R'^{(j)}_i$ from the corresponding PE's of the MAIN

and RESIDUE Units.

These inputs are shown in Figure (4.3).

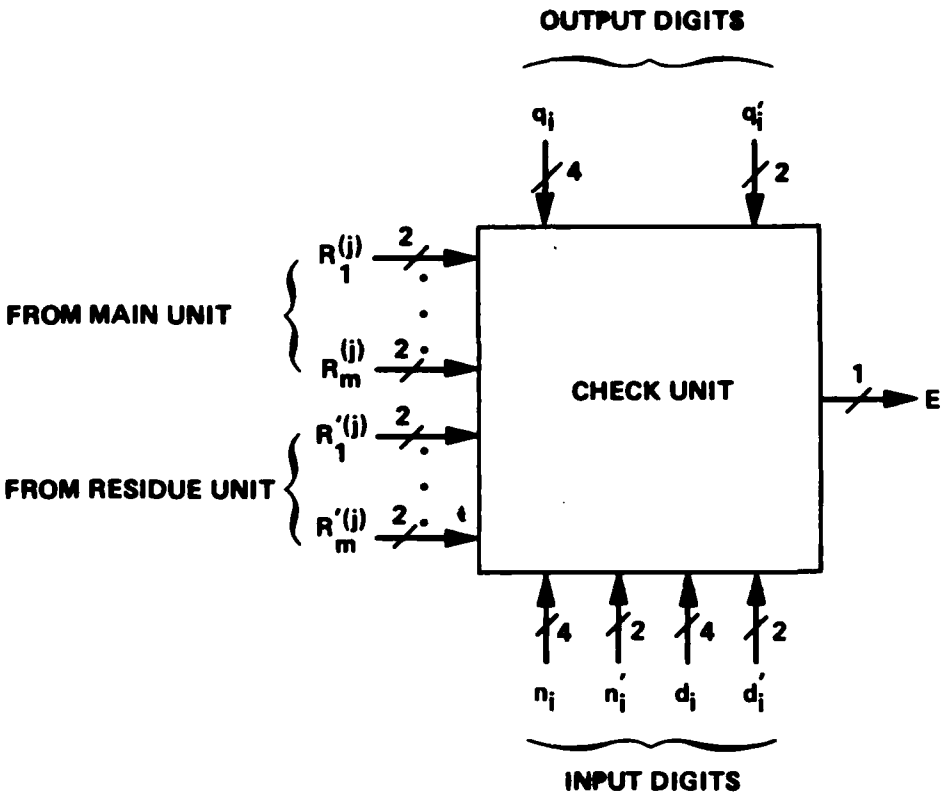


Figure (4.3)- Inputs to The CHECK Unit

The numbers shown on the block diagram belong to the case where r=10, r'=4 and A=3. In this case:

Total Number of Inputs =4m +18 (bits)

Inside the CHECK Unit $R_i^{(j)}$'s are added to generate $|P_j|_A$ and $R'^{(j)}_i$'s are added to generate $|P'_j|_A$. Therefore,

by looking at the algorithm run by the CHECK Unit (Algorithm "DETECT DIVIDE" in Chapter 3) the block diagram of this unit will be obtained as shown in Figure (4.4).

In what follows the hardware implementation of each of the components of the CHECK Unit will be considered.

Block No. 1

This block adds $n'_{j+\delta}$ [α bits] to Y_{j-1}[α bits] and obtains the residue of the result. Therefore:

$$\begin{cases} \text{time required} = t_1 = t_M \\ \text{ROM needed} = 2^{2\alpha} * \alpha \quad \text{(bits)} \end{cases}$$

Blocks No. 2 and 4

These two blocks are similar and their hardware requirements are:

$$\begin{cases} t_2 = t_4 = 2t_M \\ M_2 = M_4 = [2^{k+1} + 2^{2\alpha}]\alpha \quad \text{(bits)} \end{cases}$$

Block No. 3

This block adds q'_j (k'+1) to X'_{j-1} [α bits] and obtains the residue of the result.

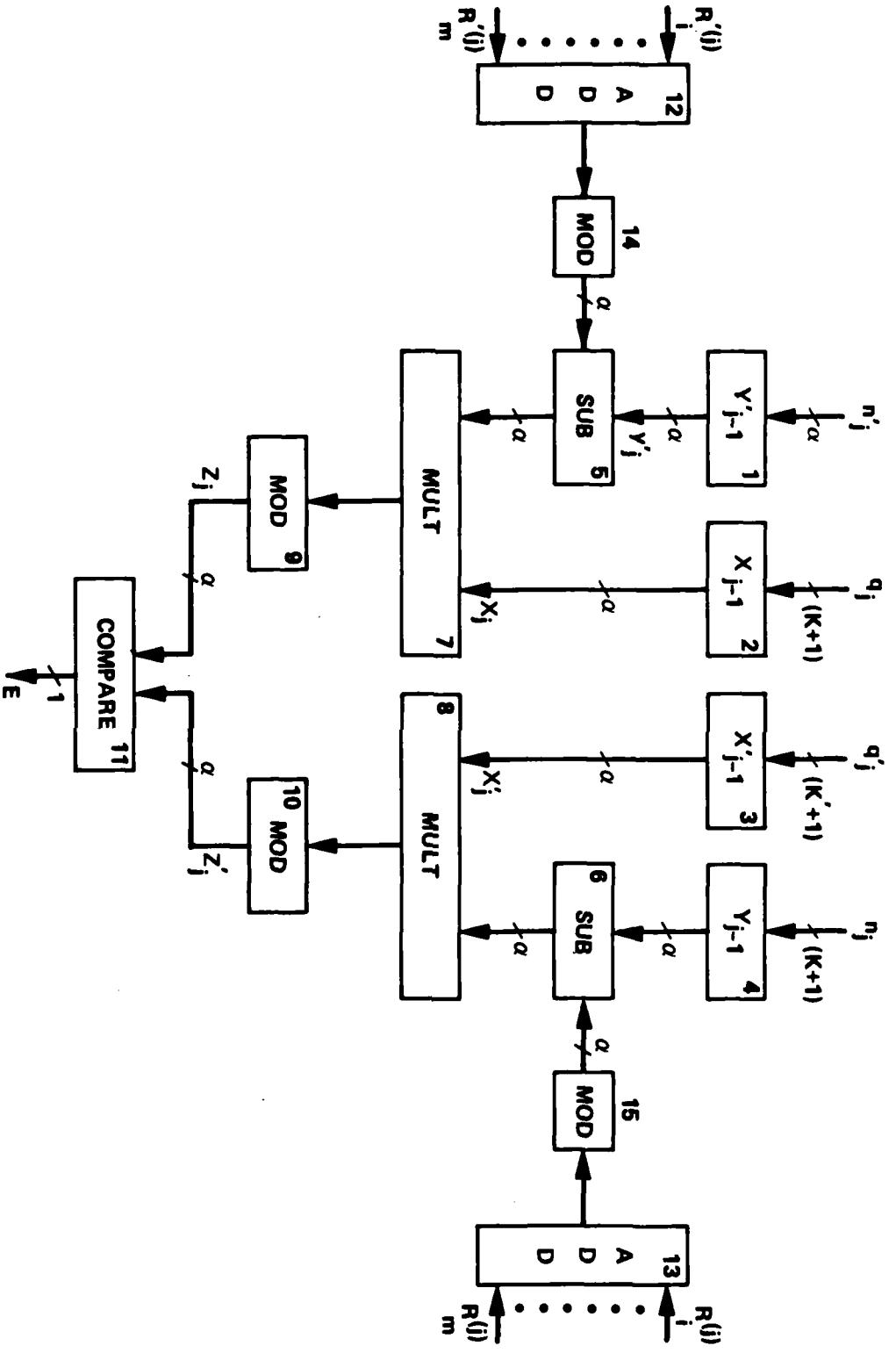


Figure 4.4 – An Implementation of The CHECK Unit

$$\begin{cases} t_3 = 2t_M \\ M_3 = [2^{k'+1} + 2^{2\alpha}] * \alpha \quad \text{(bits)} \end{cases}$$

Blocks No. 5 and 6

Block 5(6) subtracts $|P_j|_A$ ($|P'_j|_A$) from Y_j (Y'_j) and obtains the residue of the result. Therefore:

$$\begin{cases} M_5 = M_6 = 2^{2\alpha} * \alpha \quad \text{(bits)} \\ t_5 = t_6 = t_M \end{cases}$$

Blocks No. 7, 8, 9 and 10

Block 7(8) multiplies two digits of α bits each. Block 9(10) obtains the residue of the result. Therefore, the combination of blocks 7 and 9 (8 and 10) requires the following amount of hardware:

$$\begin{cases} M_7 + M_9 = M_8 + M_{10} = 2^{2\alpha} * \alpha \quad \text{(bits)} \\ t_7 + t_9 = t_8 + t_{10} = t_M \end{cases}$$

Block No. 11

This is a simple comparator which compares two residues of α bits each. This block can be implemented by a level of exclusive OR's followed by an OR gate. Therefore, we need α XOR's and one large OR gate. Assuming 3 gates per XOR and two gate delay for each we find:

$$\begin{cases} G_{11}=3\alpha+1 \\ t_{11}=3\delta_g \end{cases}$$

Blocks No. 12,13,14 and 15

Block 12(13) adds m operands of α bits each. Block 14 (15) obtains the residue of the result. The combination of these two blocks is realized by levels of ROM devices as shown in Figure (4.2). The number of levels required is:

$$L=\lceil \log_2 m \rceil \tag{4.19}$$

Figure (4.5) depicts this organization when m=8 and A=3.

The number of modules required for this process is:

$$\text{No. of Modules} = 1+2+2^2+.....+\frac{m}{2} = m-1$$

Therefore, the memory required is :

$$M_{12}+M_{14}=M_{13}+M_{15}=(m-1)*2^{2\alpha}*\alpha \tag{4.20}$$

From (4.19) the time required is:

$$t_{12}+t_{14}=t_{13}+t_{15}=L*t_M=\lceil \log_2 m \rceil t_M \tag{4.21}$$

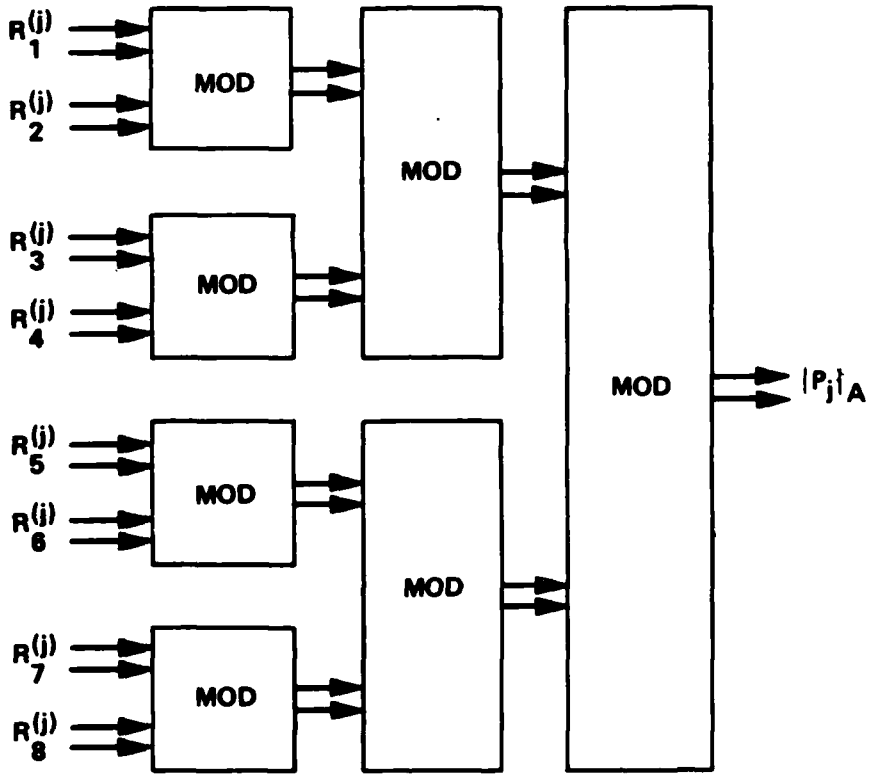


Figure 4.5 – Combination of Blocks No. 12 and 14

4.2 Cost of The Residue-Coded Division Unit

We assume that the number of gates and the amount of memory required for a unit is an indication of its cost. Therefore, in this section the overall gate complexity of the residue-coded on-line division unit will be considered.

4.2.1 Cost of The MAIN Unit

The number of gates required for a Processing Element of the MAIN Unit is (see Appendix-A, Eq. A.30):

$$G_{PE}=64k^2+157k+123 \tag{4.22}$$

On the other hand incorporation of error detection schemes requires addition of extra hardware to each Processing Element. This extra hardware is in the form of a ROM module added to each PE. The capacity of this ROM (M_{EC-PE}) is given by Eq. (4.14). Therefore the hardware requirements of each residue-coded Processing Element is:

$$\begin{cases} G_{EC-PE}=G_{PE}=64k^2+157k+123 \\ M_{EC-PE}=M_R=\alpha\, 2^k[5-2^{-\alpha+1}] + \alpha[8k+2^{\alpha^2+2}-2^{\alpha+1}] \\ \qquad\qquad +\alpha\, 2^{2\alpha}\,\frac{k+4}{2} +32k \quad \text{(bits)} \end{cases} \tag{4.23}$$

Since the MAIN Unit is composed of m PE's, gate and memory required for this unit is:

$$\begin{cases} G_{EC-MAIN} = mG_{EC-PE} \\ M_{EC-MAIN} = mM_{EC-PE} \end{cases} \qquad (4.24)$$

4.2.2 Cost of The RESIDUE Unit

The only difference between the MAIN and the RESIDUE Units is in their radices of implementation (r and r'). Therefore the total cost of the RESIDUE will be given by an equation similar to (4.24) . That is:

$$\begin{cases} G_{EC-RES} = m(64k'^2+157k'+123) \\ M_{EC-RES} = m\{\alpha\, 2^{k'}[5-2^{-\alpha+1}]+\alpha[8k'+2^{\alpha^2+2}-2^{\alpha+1}] \\ \qquad +\alpha\, 2^{2\alpha}\,\frac{k'+4}{2}+32k'\} \quad \text{(bits)} \end{cases} \qquad (4.25)$$

4.2.3 Cost of The CHECK Unit

Hardware requirements of the CHECK Unit can be obtained by adding the hardware needed for each of its components. Looking back to Section 4.1.3 we get:

$$\begin{cases} M_{CHECK} = \sum_{i=1}^{15} M_i \\ G_{CHECK} = G_{11} \\ t_{CHECK} = MAX(t_{12}+t_{14}, t_4)+t_5+t_7+t_9+t_{11} \end{cases} \qquad (4.26)$$

Using the values from Section 4.1.3 into (4.26) we get:

$$\begin{cases} M_{CHECK}=2\alpha\,[(m+3)2^{2\alpha}+2^{k+1}+2^{k'}] \\ G_{CHECK}=3\alpha+1 \\ t_{CHECK}=t_M\left[2+\left\lceil\log_2 m\right\rceil\right]+3\delta_g \end{cases} \qquad (4.27)$$

REMARKS:

1. Time required by the CHECK Unit is independent of the radix used in the MAIN Unit and only depends on the required precision.

2. In deriving Eq. (4.27) we have not put any restriction on the check modulus (A). Therefore, this equation is valid for all A's. But if we assume that A is a low-cost modulus ($A=2^{\alpha}-1$) then this results in simplification of the units and a corresponding decrease in cost and delay.

Table (4.1) illustrates Equations (4.24), (4.25), (4.26) and the overall hardware requirements of a residue-coded division unit with respect to r and m.

m	r	MAIN UNIT			RESIDUE UNIT			CHECK UNIT			RC-DIVIDE UNIT (TOTAL)		
		$G_{EC\text{-}MAIN}$	$M_{EC\text{-}MAIN}$	$t^{(\delta g)}_{EC\text{-}MAIN}$	$G_{EC\text{-}RES}$	$M_{EC\text{-}RES}$	$t^{(\delta g)}_{EC\text{-}RES}$	G_{CHECK}	M_{CHECK}	$t^{(\delta g)}_{CHECK}$	$G_{RC\text{-}DIVIDE}$	$M_{RC\text{-}DIVIDE}$	$T^{(\delta g)}_{RC\text{-}STEP}$
m=8	4	5544	2656	54	5544	2656	54	7	752	23	11095	6064	54
	16	14200	4608	77	5544	2656	54	7	848	23	19751	8112	77
	64	26952	9088	93	5544	2656	54	7	1232	23	32503	12976	93
	256	43800	23936	116	5544	2656	54	7	2768	23	49351	29360	116
m=16	4	11088	5312	54	11088	5312	54	7	1264	27	22183	11888	54
	16	28400	9216	77	11088	5312	54	7	1360	27	39495	15792	77
	64	53904	18176	93	11088	5312	54	7	1744	27	64999	25232	93
	256	87600	47872	116	11088	5312	54	7	3280	27	98695	56464	116
m=32	4	22176	10624	54	22176	10624	54	7	2288	31	44359	23536	54
	16	56800	18432	77	22176	10624	54	7	2384	31	78983	31440	77
	64	107808	36352	93	22176	10624	54	7	2768	31	127 K	49744	93
	256	171 K	95744	116	22176	10624	54	7	4304	31	192 K	108 K	116
m=64	4	44352	21248	54	44352	21248	54	7	4336	35	88711	46832	54
	16	110 K	36864	77	44352	21248	54	7	4432	35	154 K	62544	77
	64	210 K	72704	93	44352	21248	54	7	4816	35	253 K	98768	93
	256	342 K	187 K	116	44352	21248	54	7	6352	35	385 K	213 K	116

Table 4.1 – Hardware and Time Requirements of the Residue-Coded Divide Unit When $r' = 4 (\rho' = 3)$ and $A = 3 (\alpha = 2)$

CHAPTER 5

PERFORMANCE EVALUATION

5.1 Code Performance

The purpose of this chapter is to analyze the effect of imposing error codes on the existing on-line algorithms. The economic feasibility of arithmetic error codes in a computer system depends on their cost and effectiveness with respect to the set of arithmetic algorithms and their speed requirements. The choice of a specific code from the available alternatives further depends on their relative cost and effectiveness values.

Arithmetic error codes are of special interest in the design of fault-tolerant computer systems, since they serve to detect (and correct) errors in the results produced by arithmetic processors as well as the errors which have been caused by faulty transmission or storage. The same encoding is applicable throughout the entire computing system to provide concurrent diagnosis, i.e., error detection which occurs concurrently with the operation of the computer. Real time detection of transient and permanent faults is obtained without a duplication of arithmetic processors. This chapter presents the result of an investigation of the cost,

speed, interconnection requirements and effectiveness of arithmetic error codes in on-line networks. We focus our attention on the residue and AN coded division units. The results obtained can be extended to other on-line operations with the corresponding modifications.

5.1.1 Hardware and Interconnection Requirements

We define the "perfect unit" to be a unit in which logic faults do not occur. The specified set of arithmetic algorithms is carried out with prescribed speed and without errors. For a given algorithm, word length, and number representation system of the perfect unit the introduction of any code will result in changes that represent the cost of the code. The components of the cost are discussed below in general terms applicable to all arithmetic error codes [AVI 71].

1) Word length: The encoding introduces redundant bits in the number representation. A proportional hardware increase takes place in storage arrays, data paths, and processor units. The increase is expressed as a percentage of the perfect design. "Complete duplication" (100 percent increase) is the encoding which serves as the limiting case. In residue encoding, the residue of each digit with respect to A is attached to it and should be carried along with the corresponding digit. Assuming that the operands and the

results belong to a redundant number system we have:

$$x_i \in \{-\rho, \ldots\ldots, -1, 0, 1, \ldots\ldots, \rho\} \qquad (r-1) \geq \rho \geq r/2$$

The corresponding operands in the RESIDUE unit belong to the set:

$$x'_i \in \{0, 1, 2, \ldots\ldots, (A-1)\} \qquad A < r'$$

The number of bits required to represent x_i is:

$$n = \lceil \log_2 2\rho \rceil$$

Similarly, the number of bits required to represent x'_i is:

$$n' = \lceil \log_2 A \rceil$$

Therefore, all the data paths should be increased by the factor of n'/n.

$$n'/n = \lceil \log_2 A \rceil / \lceil \log_2 2\rho \rceil \simeq \lceil \log_{2\rho} A \rceil \qquad (5.1)$$

Also all the storage requirements of the units will increase by the same factor.

When using AN codes, digits of all the operands belong to the following set:

$$x'_i \in \{-A\rho, \ldots\ldots, -A, 0, A, \ldots\ldots, A\rho\}$$

Therefore, the total number of bits required is:

$$n' = \lceil \log_2 2A\rho \rceil$$

and the factor by which the word length increases is:

$$\Delta=(n'-n)/n \qquad (5.2)$$

For example if:

$r=10$, $\rho=5$ and $A=7$

Then:

$n=4$ and $n'=7$

This results in:

$\Delta=75\%$

2) The Checking Algorithm: This test the code validity of every incoming operand and every result of an instruction. A correcting operation follows when an error-correcting code is used. The cost of the checking algorithm has two interrelated components: the hardware complexity and the time required by checking. The complete duplication case requires only bit by bit comparison; other codes require more hardware and time. Provisions for fault detection in the checking hardware itself are needed and add to the cost.

In the residue scheme, the checking is done by the CHECK unit and consists of comparing the outputs of the RESIDUE and the MAIN units. This operation is performed by the "DETECT DIVIDE" algorithm mentioned in Chapter 3. Therefore, the only extra hardware we require for checking algorithm is the CHECK unit. A sample block diagram of this unit is shown in Section 4.1.3 of this thesis. By referring to this figure, we note that the hardware required to imple-

ment such unit is not complex at all. Also, in the next section we prove that the checking procedure does not introduces any delay into the operation of the on-line units.

3) The Arithmetic Algorithms: An encoding usually requires a more complex arithmetic operation than the perfect computer. This cost is expressed by the incremental time and hardware required by new algorithm. As was mentioned earlier in this thesis, the algorithms used by the error coded units are not different from those used by the ordinary units. Therefore, imposing error codes on on-line units does not add any cost of this type. Also, note that we do not require new algorithms for the residue units. The algorithms "RESIDUE OP" are exactly the same as "MAIN OP" algorithms, but they are run on the residue operands.

5.1.2 Time Requirements

Introduction of error detection schemes into the operation of an on-line divide unit results in increase of the basic recursion step time (T_{STEP}). This increase in time is due to the following two factors:

1. Time required to compute $R_i^{(j)}$'s and $R'^{(j)}_i$'s in the i-th on-line Processing Element (t_r).

2. Time required by the CHECK Unit (Eq. 4.27)

Time requirements of the MAIN on-line divide unit is considered in Appendix B of this thesis. From Eq. (B.15) in this appendix T_{STEP} is:

$$T_{STEP}=t_s+t_R=t_s+t_{PE}=\{8k+7\lceil\log_2(k+1)\rceil+24\}\delta_g \qquad (5.3)$$

Adding 1 and 2 above to this equation, the basic recursion step time for the residue-coded units ($T_{RC-STEP}$) will be:

$$T_{RC-STEP}=t_s+t_{PE}+t_r+t_{CHECK} \qquad (5.4)$$

The process of obtaining $R_i^{(j)}$'s can be started as soon as the registers TP1, TP2, TA and RW are loaded with the correct values. Having this in mind the graph representation of $T_{RC-STEP}$ will be obtained as shown in Figure (5.1).

As this diagram indicates, while the CHECK Unit is examining the results of the j-th step, MAIN and RESIDUE Units are in the (j+1)-th step. This is possible because for all values of the radix (r) the following inequality is satisfied:

$$t_s+t_{PE} > t_r+t_{CHECK} \quad \text{for all r's} \qquad (5.5)$$

This means that the results of the j-th step can be checked by the CHECK Unit while (j+1)-th step is in progress. Therefore, there is no time penalty involved in introducing the check procedure. That is:

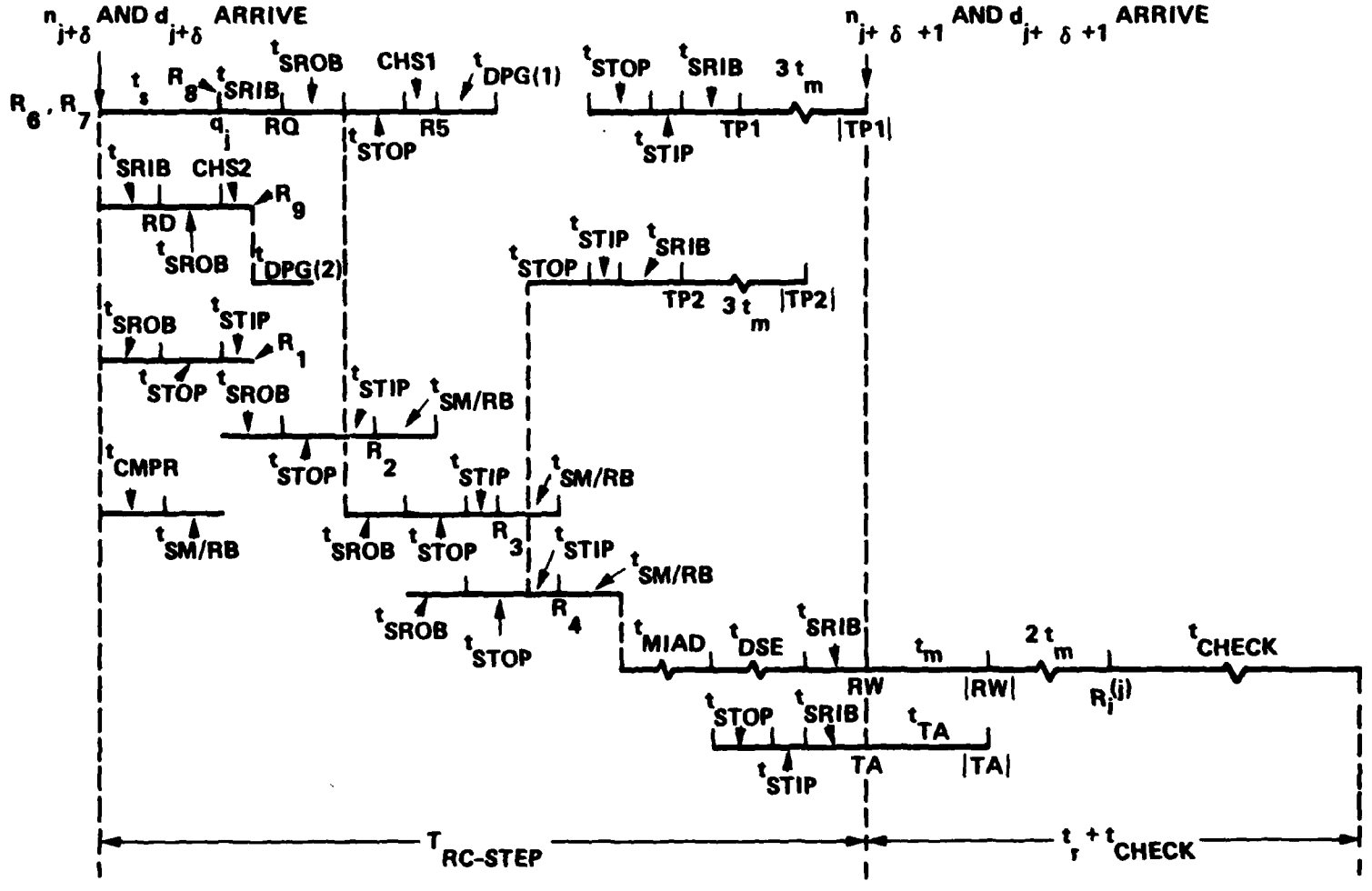


Figure 5.1 – Graph Representation of $T_{RC\text{-}STEP}$

$$T_{RC-STEP}=T_{STEP}=\{8k+7\lceil\log_2(k+1)\rceil+24\}\delta_g \qquad (5.6)$$

The values of $T_{RC-STEP}$ for various values of k have been shown in Table (4.1). Clearly:

$$t_{EC-MAIN}=T_{RC-STEP}=T_{STEP} \qquad (5.7)$$

5.1.3 Cost and Delay Comparison

As was mentioned earlier, there is no time penalty involved in using the error-detection schemes. The only penalty that we have to pay is the extra hardware needed for the RESIDUE and CHECK Units. In this section a comparison is made between the cost of the residue-coded and ordinary on-line division units. Table (5.1) has been obtained from Table (4.1) and shows the gate and memory requirements of the two units.

Δ_G is defined as:

$$\Delta_G=\frac{G_{RC-DIVIDE}-G_{DIVIDE}}{G_{DIVIDE}}*100\% \qquad (5.8)$$

Using this table the following results are obtained:

REMARKS:

1. Δ_G is not sensitive to the number of Processing Elements (m).
2. Δ_G decreases as the radix of the MAIN Unit (r) increases.

m	r	RC-DIVIDE UNIT		DIVIDE UNIT	$\Delta_{G(\%)}$
		$G_{RC-DIVIDE}$	$M_{RC-DIVIDE}$	G_{DIVIDE}	
m=8	4	11095	6064	5544	100.1
	16	19751	8112	14200	39.1
	64	32503	12976	26952	20.6
	256	49351	29360	43800	12.7
m=16	4	22183	11888	11088	100.1
	16	39495	15792	28400	39.1
	64	64999	25232	53904	20.6
	256	98695	56464	87600	12.7
m=32	4	44359	23536	22176	100.0
	16	78983	31440	56800	39.1
	64	127 K	49744	107808	20.6
	256	192 K	108 K	171 K	12.3
m=64	4	88711	46832	44352	100.0
	16	154 K	62544	110 K	40.0
	64	253 K	98768	210 K	20.5
	256	385 K	213 K	342 K	12.6

Table 5.1 – Comparison of The Gate and Memory Requirements of The RC-Divide and Divide Units

This implies that it is beneficial to use higher radices for the MAIN Unit. Also it is clear that in order for the design to be economically feasible, the radix of the MAIN Unit should be greater than the radix of the RESIDUE Unit (r').

5.2 Code Effectiveness

An arithmetic error occurs when a logic fault causes the change of one or more digits in the result of an algorithm. A logic fault is defined to be the deviation of one or more logic variables from the values specified in the perfect design. Logic faults differ in their duration, extent, and nature of the deviation from perfect values. The effectiveness of an arithmetic error code in a computer may be expressed in two forms: as a direct value effectiveness, and as a design-dependent fault effectiveness [AVI 71].

1) Value Effectiveness: The most direct measure of effectiveness is the listing of the error values that will be detected or corrected when the code is used. These values are determined by the properties of the code and are independent of the logic structure of the computer in which the code will be used. Value effectiveness for 100 percent detection (or correction) of some class of error values has been the main measure of arithmetic codes. For example, single error detection (or correction) is said to occur when all (100 percent) errors of value

$$\pm cr^i \qquad 0<c<r \qquad 0<i<m-1$$

are detected (or corrected) in an m-digit, radix-r number. There is no direct reference for algorithms or their implementation. Codes with value effectiveness of less than 100 percent detection are useful when their cost is low and when

other means of fault tolerance supplement the codes in a computer.

2) Fault Effectiveness: The purpose of arithmetic error codes in digital systems is to detect the occurrence of logic faults. The detection enables the system to initiate corrective action (error correction, diagnosis, program restart, etc.). In order to assess the effectiveness of fault detection, the value effectiveness of a code must be translated into a measure of fault effectiveness for one or more specified types of logic faults. The translation is performed separately for every algorithm and requires an error table for every type of fault. The error table is generated from the description of the logic implementation of the algorithms. The specified fault is applied to every logic circuit which is used by the algorithm. Every application yields an error value (or a set of error values) by which the fault will change the perfect value of the result to the actual (incorrect) value. The error table lists all error values together with their relative frequencies of occurrence during the compilation of the error table. A comparison of the error table with the detectable error values of the given code shows which entries of the error table are not detectable. Therefore, the fault effectiveness of a code with respect to the given algorithm and the specified fault is the percentage of all occurrences of this fault which will be detected (or corrected) when the given code is em-

ployed. Less than 100 percent fault-effective codes are of interest when their cost is low, because other methods of fault tolerance can be used of reinforce the code. If the fault-effectiveness for an algorithm and a given fault is not sufficient, it may be improved by redesigning the implementation of the algorithm to eliminate some or all of the undetectable entries of the error table [AVI 71].

5.2.1 Error Detection Analysis for Residue Encoding

In this and the following sections we focus our attention on the error-coded on-line divide unit. Error detection capabilities of this unit when residue encoding is used will be considered in this section and the following section addresses the same problem when AN codes are used.

By referring to the block diagrams of the MAIN and the RESIDUE units shown in Chapter 4, the logic faults that can happen in the error-coded units can be divided into two parts:

1) Logic faults that occur in the SELECTION blocks of the MAIN and the RESIDUE units. These faults result in the selection of incorrect quotient digits (q_j and q'_j).

2) Faults in the other parts of the units including faults in Multi-input Redundant Adders, Operand Registers and the digit transfer operation.

The proposed residue scheme cannot detect the first category of errors, i.e., errors in the SELECTION Units. The reason is that, errors in q_j and q'_j are compensated by the resulting errors in the corresponding partial remainders P_j and P'_j. This is due to the step 4 of the "MAIN DIVIDE" and "RESIDUE DIVIDE" algorithms. But this type of error can easily be detected by the range test of the corresponding partial remainders P_j and P'_j. The following theorem proves this claim.

Theorem (1): Any deviations of the selected quotient digits from the correct value will result in a partial remainder which is out of bounds.

Proof

The maximum and minimum values of the j-th partial remainder (P_j) with non-redundant operands have been derived in [GOR 80] and are:

$$kD_j - r^{-\delta} \geq P_j \geq -kD_j + kr^{-\delta} \tag{5.9}$$

Similarly, P'_j is bounded by:

$$k'D'_j - r'^{-\delta} \geq P'_j \geq -k'D'_j + k'r'^{-\delta} \tag{5.10}$$

An error in either SELECTION units may increase (or decrease) the value of the j-th quotient digit $q_j=i$ ($q'_j=i'$) by the amount of E (E'). Figure (5.2) shows a rP_j-D_j plot and the corresponding $q_j=i$ selection regions of the MAIN

Unit.

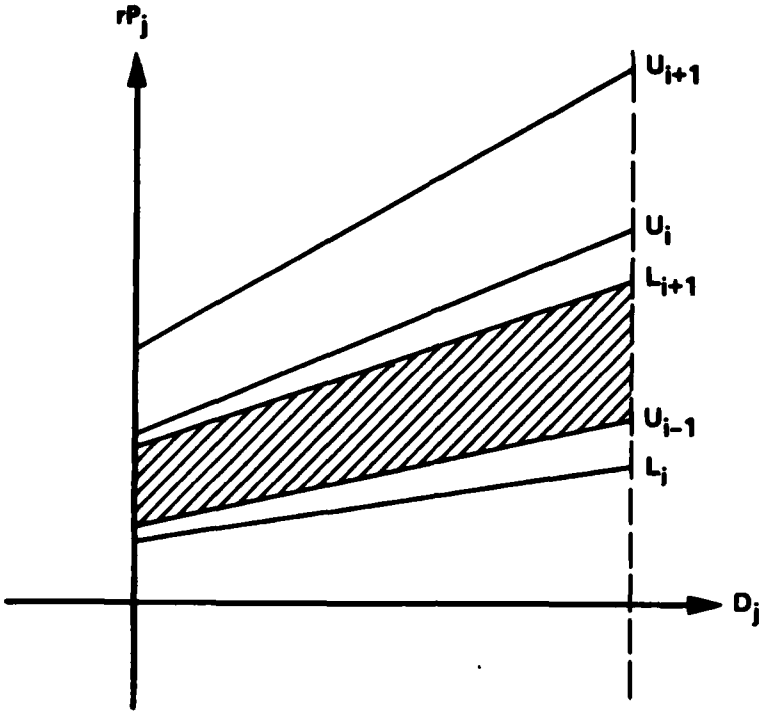


Figure (5.2)- A Sample rP_j-D_j Plot

Assume that the (j-1)-th partial remainder is in the following range (shaded area).

$$L_{i+1} \geq rP_{j-1} \geq U_{i-1} \tag{5.11}$$

It is clear that the only acceptable value of the j-th quotient digit is:

$$q_j = \text{SELECT}(rP_{j-1}, D_{j-1}) = i \tag{5.12}$$

Now assume that due to an error in the SELECTION Unit, the actual (incorrect) value of q_j is:

$$q_j^* = i+E \tag{5.13}$$

The following two equations have been derived in [GOR 80]:

$$\begin{cases} U_i = (i+k)D_j - r^{-\delta+1} \\ L_i = (i-k)D_j + kr^{-\delta+1} \end{cases} \tag{5.14}$$

Inserting (5.14) into (5.11) we get:

$$(i+1-k)D_{j-1} + kr^{-\delta+1} \geq rp_{j-1} \geq (k+i-1)D_{j-1} - r^{-\delta+1} \tag{5.15}$$

Using these maximum and minimum values of rP_{j-1} in the basic recursion step (Eq. 3.12), the bounds on P_j are obtained:

$$MAX(P_j) = (i+1-k)D_{j-1} + kr^{-\delta+1} - (i+E)D_j + (r-1)r^{-\delta} \tag{5.16}$$

$$MIN(P_j) = (k+i-1)D_{j-1} - r^{-\delta+1} - (i+E)D_j - k(r-1)r^{-\delta} \tag{5.17}$$

Assuming $D_j \tilde{\sim} D_{j-1}$, (5.16) and (5.17) reduce to:

$$MAX(P_j) = (1-k-E)D_j + kr^{-\delta+1} + r^{-\delta+1} - r^{-\delta} \tag{5.18}$$

$$MIN(P_j) = (k-1-E)D_j - r^{-\delta+1} - kr^{-\delta+1} + kr^{-\delta} \tag{5.19}$$

The allowable values of δ are [GOR 80]:

$$r^{-\delta+1} \leq \frac{2k-1}{k+1} D_j \tag{5.20}$$

Inserting (5.20) in (5.18) and (5.19) we get:

$$MAX(P_j) = (k-E)D_j - r^{-\delta} \tag{5.21}$$

$$MIN(P_j) = -(k+E)D_j + kr^{-\delta} \tag{5.22}$$

Clearly, when E=0 these bounds should be equal to those obtained previously (see Eq. 5.9). But, when $E \neq 0$, it is easy

to prove that the resulting P_j is out of bounds.

Case 1) E is Positive:

Assume:

$$MAX(P_j)=(k-E)D_j-r^{-\delta}\leq \frac{L_{-\rho}}{r} \qquad (5.23)$$

Inserting the value of $L_{-\rho}$ from (5.9) in (5.23) we get:

$$(2k-E)D_j\leq(1+k)r^{-\delta}$$

Replacing δ from (5.20) we get:

$$E\geq\frac{2\rho+1}{r} \qquad (5.24)$$

ρ is defined to be in the following range:

$$r-1\geq \rho \geq\frac{r}{2}$$

Using this relation in (5.24) results in:

$$E\geq 1+\frac{1}{r} \qquad (5.25)$$

This means if $E\geq 2$ then P_j will be out of the correct bounds. Note that the given derivation was for the worst case and usually the smallest possible value of E (E=1) will generate a partial remainder which is well out of bounds. This can be an indication of an error.

Case 2) E is Negative:

Similar analysis follows in this case and the result is:

$$E \leq -1 - \frac{1}{r} \tag{5.26}$$

Therefore, in order to detect errors in the SELECTION units, we can compare the magnitude of the resulting partial remainders with the maximum and the minimum allowable values. If this value is within the range then no error has occurred. But, if it is out of bounds and the CHECK unit does not indicate an error, then the SELECTION unit of the corresponding module is malfunctioning.

The second type of faults that we are concerned about are those that do not affect the selection function. They may occur in other parts of the units including, the registers which hold the operands, multi-input redundant adders, carry generation blocks and partial remainder registers. These errors are detectable by the proposed residue scheme as long as the value of the error is not divisible by the check constant A. Referring to algorithm "DETECT DIVIDE" this means if $Z_j \neq Z'_j$ for $j=1,2,\ldots,m$.

Since the MAIN and the RESIDUE units are totally separate, compensation of errors does not happen. But errors will remain undetectable if they occur in only one unit and $|\epsilon|_A=0$ or in two units and $|\epsilon_1 - \epsilon_2|_A=0$.

As an example assume an error in the multi-input redundant adder of the MAIN Unit changes the perfect value of partial remainder P_j to an actual (incorrect) value P_j^*.

Therefore, referring to the algorithm "DETECT DIVIDE" we have:

$$Z'^{*}_{j}=|X'_{j}*(Y_{j}-|P^{*}_{j}|_{A})|_{A} \tag{5.27}$$

$$Z'_{j}=|X'_{j}*(Y_{j}-|P_{j}|_{A})|_{A} \tag{5.28}$$

$$Z_{j}=|X_{j}*(Y'_{j}-|P'_{j}|_{A})|_{A} \tag{5.29}$$

Subtracting (5.27) from (5.29) and taking the residues of both sides with respect to A we get:

$$|Z_{j}-Z'^{*}_{j}|_{A}=|Z'_{j}-Z'^{*}_{j}|_{A}=|X'_{j}*(|P^{*}_{j}-P_{j}|_{A})|_{A} \tag{5.30}$$

The difference between P^{*}_{j} and P_{j} is the error (ϵ).

$$\epsilon=P^{*}_{j}-P_{j}$$

Inserting this in (5.30) we get:

$$|Z_{j}-Z'^{*}_{j}|_{A}=|X'_{j}*\epsilon|_{A}=\big||X'_{j}|_{A}*|\epsilon|_{A}\big|_{A} \tag{5.31}$$

Therefore, the error will go undetected if and only if:

$$|X'_{j}|_{A}=0$$

or

$$|\epsilon|_{A}=0$$

Because, when $Z_{j}=Z'^{*}_{j}$ step 3 of the algorithm "DETECT DIVIDE" cannot catch the error.

For single error we have:

$\epsilon=\pm Cr^{-j}$ $\rho \geq C \geq 1$ MAIN UNIT

$$\epsilon = \pm C' r'^{-j} \quad \rho' \geq C' \geq 1 \quad \text{RESIDUE UNIT}$$

Assuming $|r|_A = |r'|_A = 1$ we get:

$$|\epsilon|_A = |C|_A$$

$$|\epsilon'|_A = |C'|_A$$

Therefore, single digit errors will go undetected if:

$$|C|_A = 0 \tag{5.32}$$

$$|C'|_A = 0 \tag{5.33}$$

But for the RESIDUE Unit the following relation is satisfied (Section 3.1.3).

$$C' \leq \rho' \leq A-1 \leq r'-1 \tag{5.34}$$

From (5.34) we deduce that:

$$C' < A$$

Therefore, (5.33) can never be satisfied unless C'=0. This proves that all single digit error in the RESIDUE Unit will be detected by the proposed scheme. Similar errors in the MAIN Unit may, in some cases, go undetected.

Assuming that each radix r(r') digit is shown by $\lceil \log_2 r \rceil$ ($\lceil \log_2 r' \rceil$) bits inside the machine, all single bit errors are detectable as long as A is a low-cost modulus. An Example of the error detection capabilities of the residue encoding is shown in Section (3.1.4).

5.2.2 Error Detection/Correction Analysis for AN Encoding

When AN codes are used in an on-line unit, the checking procedure is simply finding the residue of each single digit of the results, as they are generated, with respect to the check modulus A. If the operation of the AN coded unit is fault-free, then all these residues should be equal to zero. A non-zero residue is an indication of the error. In Chapter 3 of this thesis we depicted the block diagram of an AN coded on-line divide unit. In this unit the check is performed on the quotient digit (q'_j) and the corresponding partial remainder (P'_j). Denote the number of bits required to represent n'_j, d'_j and q'_j by α', β' and γ' respectively. Looking back into equations (3.30), (3.31) and (3.32) we obtain:

$$\alpha' = \lceil \log_2 2A^2\rho'' \rceil \tag{5.35}$$

$$\beta' = \lceil \log_2 2A\rho' \rceil \tag{5.36}$$

$$\gamma' = \lceil \log_2 2A\rho \rceil \tag{5.37}$$

For instance q'_j is represented by $(\gamma'-1)$ bits for the magnitude and one bit for sign. If 2's complement number system is used, q'_j can be represented by:

$$q'_j = (x_{\gamma'-1}, x_{\gamma'-2}, \ldots, x_1, x_0) = -2^{\gamma'} x_{\gamma'-1} + \sum_{k=0}^{\gamma'-2} x_k 2^k \tag{5.38}$$

Error Detection

Single error detection requires that the minimum distance between coded numbers be 2, that is, no two coded numbers be a distance 1 apart. Thus for all permissible q'_i and q'_j (or n'_i and n'_j or d'_i and d'_j)

$$q'_i - q'_j = A(q_i - q_j) \neq 2^k$$

This can be assured by choosing A to be odd. The choice A=3 will detect any single error in the binary representation of the operands and the results. Notice that this detection of single errors does not depend on α', β' and γ' and therefore does not depend on the radix of implementation (r). This means no matter how large α', β' and γ' are, only two additional bits are sufficient for detection of a single error.

Error Correction

Error correction can also be done if the distance between coded numbers is greater than 2. For single error correction d=3 is sufficient. The following theorem specifies the range of the numbers in which a single error can be corrected using the check modulus A [PET 72].

Theorem (2): For any choice of A, if N is restricted to the range

$$\frac{-1}{2}M_2(A,d-1) \leq N < \frac{1}{2}M_2(A,d) \qquad (5.39)$$

then AN code has minimum distance of at least d.

In case of division $+\rho \geq q_i \geq -\rho$ and if d=3 then using the above theorem:

$$\rho < \frac{1}{2}M_2(A,3) \qquad (5.40)$$

As an example when r=10 and ρ=5 then A=19. Therefore, if we multiply every digit of the dividend (N) and the divisor (D) by A=19 before sending them to the on-line divide unit, then a single error in each of the quotient digits (q_j) can be corrected. An example of error-detection with AN-coded operands is shown in Section (3.1.2.3).

An Example of Error Correction

Assume r=10, $\rho=\rho'=\rho''=5$ and A=19 for single error correction. From (3.30), (3.31) and (3.32) we get:

$$\begin{cases} n'_j \in \{-1085,.....,-361,0,361,.....,1805\} \\ d'_j,\ q'_j \in \{-96,-76,...,-19,0,19,...,76,96\} \end{cases}$$

and

$$y' = \lceil \log_2 2A\rho \rceil = 8 \quad \text{bits}$$

Therefore, q'_j is represented by 8 bits inside the machine:

$$q'_j = x_7 x_6,.....,x_1,x_0$$

A single error can be in any of these 8 positions and Table (5.2) shows that all these single errors are correctable once the residue of q'^{*}_{j} (incorrect value of q'_{j}) with respect of A is obtained by the CHECK Unit.

i	bit in error	$\lvert q'^{*}_{j} \rvert_A$
0	x_0	1,18
1	x_1	2,17
2	x_2	4,15
3	x_3	8,11
4	x_4	16,3
5	x_5	13,6
6	x_6	7,12
7	x_7	14,5

Table (5.2)- Single Error Correction

Since the residues shown in Table (5.2) are unique, a single error in any of the 8 positions can be corrected without ambiguity. The following is a block diagram of the CHECK Unit for this example.

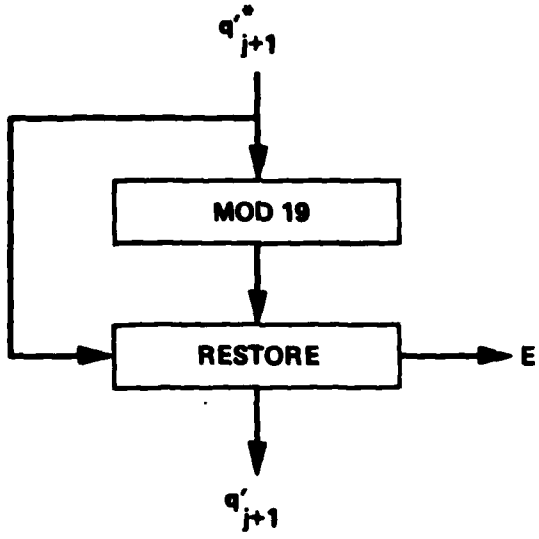


Figure (5.3)- CHECK Unit With Error-Correction Capability

CHAPTER 6

CONCLUSIONS AND SUGGESTIONS FOR FUTURE RESEARCH

In this thesis we have presented a method for detection and correction of errors in on-line arithmetic algorithms. This method is based on low-cost arithmetic error codes and encodes each digit of the operands separately. The encoded operands pass through the arithmetic unit digit-by digit, most significant digit first. The proposed algorithms are such that they preserve the codes, therefore each digit of the result must conform with its code. In this way, and depending on the code used, errors in each single digit of the result can be detected or corrected. The need for such a detection/correction scheme arises from the fact that on-line arithmetic requires relatively long sequences of operations in order to achieve speed-up over conventional arithmetic. Therefore, it is important to protect them against hardware failures. If not protected, the hardware failures could quickly contaminate large number of results in progress due to tight coupling of the steps at the digit level. By detecting errors as they occur, an effective gracefully degradable organization could be achieved. This means, error at the j-th step would lead to restriction of precision (significance) of the remaining steps but not catastrophic

termination.

We presented two methods for such a detection/correction scheme: 1) Residue encoding; 2) AN encoding. In the first method, residue of every digit of the operands with respect to a constant is attached to it and is sent to the on-line unit. Two separate processors, process the operands and their residues. The result generated can be checked for having the correct residue with respect to the same constant. In this way, we proved that a single error in each digit of the operands and the corresponding results can be detected. Also, we proved that an error in the selection of the result digits can be detected even without using the proposed residue scheme. It is interesting to note that, no new algorithms are necessary for the residue-coded operands. The algorithms are the same as those developed for ordinary operands. The only new algorithm that is needed is the one used by the detection process.

In the second approach, each digit of the operands is multipied by a constant before entering the on-line network. This code is preserved throughout the network and each digit of the final result can be checked for divisibility by the same constant. We showed that depending on the check modulus, a single error in each digit of the result can be detected or corrected.

The error-coded algorithms and a block diagram implementation of the corresponding units have been presented in this thesis. A detailed design of a digit-sliced on-line division unit was also considered. This unit was designed as a set of basic Processing Elements (PE) each of which operates on a single digit of the operands and the results. Assuming that the radix of implementation is r ($=2^k$), number of gates required for one PE has been proved to be proportional to k^2. Also, number of pins required is proportional to k. In short, we showed that the number of gates vary from 350 to 5500 for radices 2 to 256.

Processing time of a PE is also an important factor and was determined to be in the range of 39 to 116 gate delays for the aforementioned range of radices. Finally, we extended this on-line unit to encompass the residue-coded operands. We proved that the imposition of residue codes on on-line division unit increases the gate requirements by no more that 39%. The checking procedure can be overlapped with the operation of the MAIN Unit and in that sense there is no time penalty for introducing error-codes into the on-line division unit.

There remain several areas of interest that need further research such as the extension of this work to other functions such as logarithmic, trigonometric, and exponential. It is apparent that the E-method [ERC 75] is a good

candidate for such an extension. It is believed that the detection/correction procedure outlined in this thesis can be applied directly in this and similar cases. Another point of interest is the actual implementation of the on-line processing units in VLSI. Also the simulation of the proposed detection/correction schemes and experimental validation of the code effectiveness warrants further research.

REFERENCES

[ATK 68] Atkins, D.E., "Higher Radix Division Using Estimates of the Divisor and Partial Remainders", IEEE Trans. on Computers, Vol.C-17, No. 10, pp. 925-931, October 1968.

[AVI 61] Avizienis, A. "Signed Digit Number Representation for Fast Parallel Arithmetic", IRE Trans. Electron. Comput., Vol. EC-10, pp. 389-400, 1961.

[AVI 62] Avizienis, A., "On a Flexible Implementation of Digital Computer Arithmetic ", Proceeding of IFIP Congress, Munich, W. Germany, Aug 27 to Sept 1, 1962, pp. 204-211.

[AVI 65] Avizienis, A., "A Study of the Effectiveness of Fault-Detecting Codes for Binary Arithmetic," Jet Propulsion Lab., Pasadena, Calif., Tech. Rep. 32-711, Sept. 1, 1965.

[AVI 66] Avizienis, A., " Arithmetic Microsystems for The Synthesis of Function Generators", Proceeding of the IEEE, Vol. 54, No.12, Dec 1966, pp. 1910-1919.

[AVI 67] Avizienis, A., "Concurrent Diagnosis of Arithmetic Processors," in Dig. 1st Annu. IEEE. Comput. Conf., pp. 34-37, Sept. 1967.

[AVI 68] Avizienis, A., "Theory of Digital Computer Arithmetic", Notes for Engineering 225A, pp. 1-5, UCLA Computer Science Dept., 1968-1969.

[AVI 70] Avizienis, A. and Tung, C., "A Universal Arithmetic Building Element (ABE) and Design Methods for Arithmetic Processors", IEEE Transactions on Computers, Vol C-19, No. 8, August 1970, pp. 733-745.

[AVI 71] Avizienis, A., "Arithmetic Error Codes: Cost and Effectiveness Studies for Application in Digital System Design", IEEE Trans. on Comput., Vol. C-20, No. 11, Nov. 1971, pp. 1322-1331.

[AVI 73] Avizienis, A., "Arithmetic Algorithms for Error-Coded Operands", IEEE Trans. on Comput., Vol. C-22, No. 6, June 1973, pp. 567-572.

[AVI 79] Avizienis, A. et al., "Proposal for Research in Distributed Processing Systems", Submitted to the Office of Naval Research, Report No. UCLA-ENG-P-4504-N-79, June 1979.

[BOR 68] Borovec, R.T., "The Logical Design of a Class of Limited Carry-Borrow Propagation Adders", M.S. Thesis, Report 275, Department of Computer Science, University of Illinois, Urbana, August 1968.

[BRO 60] Brown, D.T., "Error Detecting and Correcting Binary Codes for Arithmetic Operations", IRE Trans. Electron. Comput., Vol.EC-9, pp. 333-337, Sept. 1960.

[CHI 72] Chien, R.T. and Hong S.J., "Error Correction in High-Speed Arithmetic", IEEE Transactions on Computers, Vol. c-21, No. 5, May 1972, pp. 433-438.

[CHU 80] Chu, M. and Ercegovac, M.D., "A Multi-Microprocessor Bit-Slice Organization for Function Evaluation", International Symposium and Micro-Computers at Asilomar, Pacific Grove, California, January 1980.

[DIA 55] Diamond, J.M., "Checking Codes for Digital Computer", Proc. IRE (Corresp.), Vol.43, pp. 487-488, April 1955.

[ERC 75] Ercegovac, M.D., "A General Method for Evaluation of Functions and Computations in a Digital Computer", Ph.D. Thesis, Report No. 750, Department of Computer Science, University of Illinois, Urbana, Augest 1975.

[ERC 78] Ercegovac, M.D., "An On-Line Square Rooting Algorithm", Proc. 4th IEEE Symp. Comput. Arithmetic, pp. 183-189, October 1978.

[ERC 80] Ercegovac, M.D. and Grnarov, A.L., "On The Performance of On-Line Arithmetic", Proc. 1980 International Conference on Parallel Processing, Harbor Springs, Michigan, August 1980.

[GOR 80] Gorji-Sinaki, A. and Ercegovac, M.D., "On-Line Division Algorithms: A Systematic Derivation", (in preparation), 1980.

[GOY 76] Goyal, Lakshmi, "A Study in the Design of an Arithmetic Element for Serial Processing in an Iterative Structure ", Ph.D. Thesis, Report 797, Department of Computer Science, University of Illinois, Urbana, May 1976.

[GRN 79] Grnarov, A.L. and Ercegovac, M.D., "An Algorithm for On-Line Normalization", UCLA Computer Science Department Quarterly, Vol.7, No.3, pp. 81-94, July 1979.

[GRN 80] Grnarov, A. and Ercegovac, M.D., "VLSI-Oriented Iterative Networks for Array Computations", 1980 International Conference on Circuits and Computers, New York, October 1980.

[HAB 70] Habibi,A. and Wintz, P.A., "Fast Multipliers", IEEE Transaction on Computers, February 1970, pp. 153-157.

[IRW 77] Irwin, M.J., "An Arithmetic Unit for On-Line Computation", Ph.D. dissertation, University of Illinois at Urbana-Champaign, report No. UIUCDCS-R-77-873.

[MAR 74] Markus, J., "Guide Book of Electronic Circuits" Mc Graw-Hill, Inc. 1974, Page 335.

[NEU 75] Neumann, P.G. and RAO, T.R.N., "Error-Correcting Codes for Byte-Organized Arithmetic Processors", IEEE Transactions on Computers, Vol. c-24, No. 3, March 1975, pp. 226-232.

[OKL 78] Oklobdzija, V.G., "An On-Line Higher Radix Square Rooting Algorithm", M.S. Thesis, Computer Science Department, University of California, Los Angeles, June 1978.

[PAR 78] Parhami, B. and Avizienis, A., "Detection of Storage Errors in Mass Memories Using Low-Cost Arithmetic Error Codes", IEEE Trans. on Computers, Vol. C-27, No. 4, April 1978, pp. 302-308.

[PET 72] Peterson, W.W and Weldon, E.J., "Error Correcting Codes", Second Edition, MIT Press, Cambridge, Mass., 1972.

[RAG 80] Raghavendra, C.S. and Ercegovac, M.D., "A Highly Functional Simulator of On-Line Algorithms", report-manual (in preparation), 1980.

[RAO 68] Rao, T.R.N., "Error-Checking Logic for Arithmetic-Type Operations of a Processor", IEEE Trans. on Comput., Vol. C-17, No. 9, Sept 1968., pp. 845-849.

[RAO 70] Rao, T.R.N., " Biresidue Error Correcting Codes for Computer Arithmetic", IEEE Trans. on Comput., Vol. C-19, No. 5, May 1970, pp. 398-402.

[RAO 72] Rao, T.R.N., "Error Correction in Adders Using Systematic Subcodes", IEEE Trans. on Comput., Vol. C-21, No. 3, March 1972, pp. 254-259.

[RAO 74] Rao, T.R.N., "Error Coding for Arithmetic Processors", Academic Press, New York, NY., 1974.

[RAO 77] Rao, T.R.N. and Reinheimer, H.J., "Fault-Tolerant Modularized Arithmetic Units", National Computer Conference, 1977, pp. 703-710.

[ROB 58] Robertson, J.E., "A New Class of Digital Division Methods", IRE Trans. on Electronic Computers, pp.88-92, September 1958.

[ROH 67] Rohatsch, Fred A., "A Study of Transformations Applicable to the Development of Limited Carry-Borrow Propagation Adders", Ph.D. Thesis, Report 226, Department of Computer Science, University of Illinois, Urbana, June 1967.

[TEX 69] Texas Instrument Incorporated, "TTL Integrated Circuits Catalog", Dallas, Texas Instruments Catalog CC201, August 1969.

[TRI 77] Trivedi, K.S. and Ercegovac, M.D., "On-Line Algorithms for Division and Multiplication" IEEE Trans. on Comput., Vol. C-26, No.7, July 1977.

[TRI 78] Trivedi, K.S. and Rusnak, J.G., "Higher Radix On-Line Division", Proc. 4th IEEE Symp. Comput. Arithmetic, pp. 164-174, October 1978.

[TUN 70] Tung, C. and Avizienis, A., "Combinational Arithmetic Systems for the Approximation of Functions", 1970 Spring Joint Computer Conf., AFIPS Proc., Vol. 36, Washington, D.C., Spartan, 1970, pp. 95-107.

[WAT 80] Watanuki, O., "A Study of On-Line Arithmetic for Highly Concurrent Execution of Numerical Algorithms", Ph.D. Dissertation (in preparation), 1980.

APPENDIX A

HARDWARE DESIGN OF AN ON-LINE DIVISION UNIT

In this appendix we consider the hardware implementation of the MAIN DIVIDE Unit. Since MAIN and RESIDUE Units have similar organizations, it is obvious that the same design can be applied to a RESIDUE Unit with minor modifications.

In order to design such a unit we assume that the on-line unit consists of a linear cascade of identical Processing Elements (PEs). Each PE is a complex logical module and contains logic to perform on-line operation under the control of the Global Control Unit (GCU).

Figure (A.1) shows the schematic organization of on-line division unit along with the GCU.
EU performs the exponent calculations.
END UNIT allows the last PE to be identical to all the other PEs as far as interface is concerned.

The PEs collectively contain the fractional parts of all active operands, one digit in each PE. Most significant digits are in PE_1 and least significant digits in PE_n. Output digits are generated by the most significant Processing

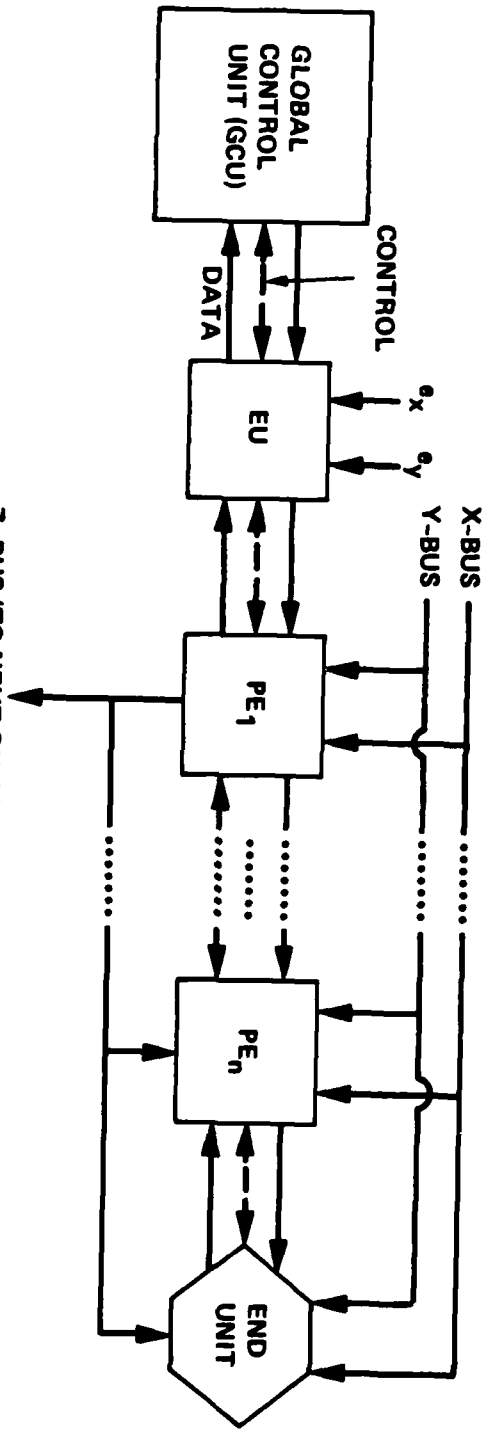


Figure A.1 – Organization of an On-Line Division Unit

Element in an on-line mode and are placed on the Z-Bus. Each output digit is stored by all PE's temporarily and at the same time reaches the next on-line unit.

After receiving the output digit and the transfer informations from the right-hand neighbor, each PE starts the computation and generates one digit of the partial remainder. Depending on this partial remainder and the truncated version of the divisor, next quotient digit is selected by PE_1 and is placed on the output bus. This operation continues until the required precision is obtained.

In order of determine the operation of each PE. we look at the basic recursion formula for on-line division (Equation 3.12):

$$P_j = rP_{j-1} - q_j D_j + n_{j+\delta} r^{-\delta} - Q_{j-1} d_{j+\delta} r^{-\delta} \qquad (A.1)$$

Assuming that each of the operands are $\underline{m}$ digits long we have:

$$P_j = \sum_{i=1}^{m} p_i^j r^{-i} \qquad (A.2)$$

$$D_j = \sum_{i=1}^{j+\delta} d_i r^{-i} \qquad (A.3)$$

p_i^j= The i-th digit of the j-th partial remainder (this digit is in PE_i).

n_i , d_i= The i-th digit of the operands (resident in PE_i).

The digits processed by PE_i in step j of the algorithm will be obtained by the following picture:

$rp_{j-1} = p_1^{j-1} * p_2^{j-1} \ldots p_{1+\delta}^{j-1} \ldots\ldots\ldots p_{i+1}^{j-1} \ldots\ldots\ldots\ldots\ldots p_n^{j-1} 0$

$D_j \quad = 0 * d_1 \ldots\ldots d_\delta \ldots\ldots\ldots\ldots d_i \ldots d_{j+\delta} \ldots 0 \ldots\ldots\ldots\ldots 0$

$n_{j+\delta} r^{-\delta} = 0 * 0 \ldots\ldots 0 n_{j+\delta} 0 \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots 0$

$Q^{j-1} r^{-\delta} = 0 * 0 \ldots\ldots 0 q_1 q_2 \ldots\ldots\ldots\ldots q_{i+1-\delta} \ldots\ldots q_{j-1} 0 \ldots\ldots\ldots 0$

* is the decimal point

Therefore, the digits processed by PE_i are obtained and (A.1) becomes:

$$p_i^{(j)} = p_{i+1}^{(j-1)} - q_j d_i + n_{j+\delta}[i=\delta] - q_{i+1-\delta} d_{j+\delta} + T_i^{(j)} - r T_{i-1}^{(j)} \qquad \text{(A.4)}$$

where $n_{j+\delta}[i=\delta]$ means $n_{j+\delta}$ will be added in PE_i only if $i=\delta$.
$T_i^{(j)}$ = transfer digit from PE_{i+1} at the j-th step
$T_{i-1}^{(j)}$ = transfer digit to PE_{i-1} at the j-th step

It is obvious that $q_{i+1-\delta}$ is zero in the Processing Element not in the following range:

$$q_{i+1-\delta} = \begin{cases} q_{i+1-\delta} & j-2+\delta \geq i \geq \delta \\ 0 & \text{otherwise} \end{cases} \qquad \text{(A.5)}$$

Using Eq. (A.4) the following picture for the on-line divide unit will be obtained (Figure A.2).

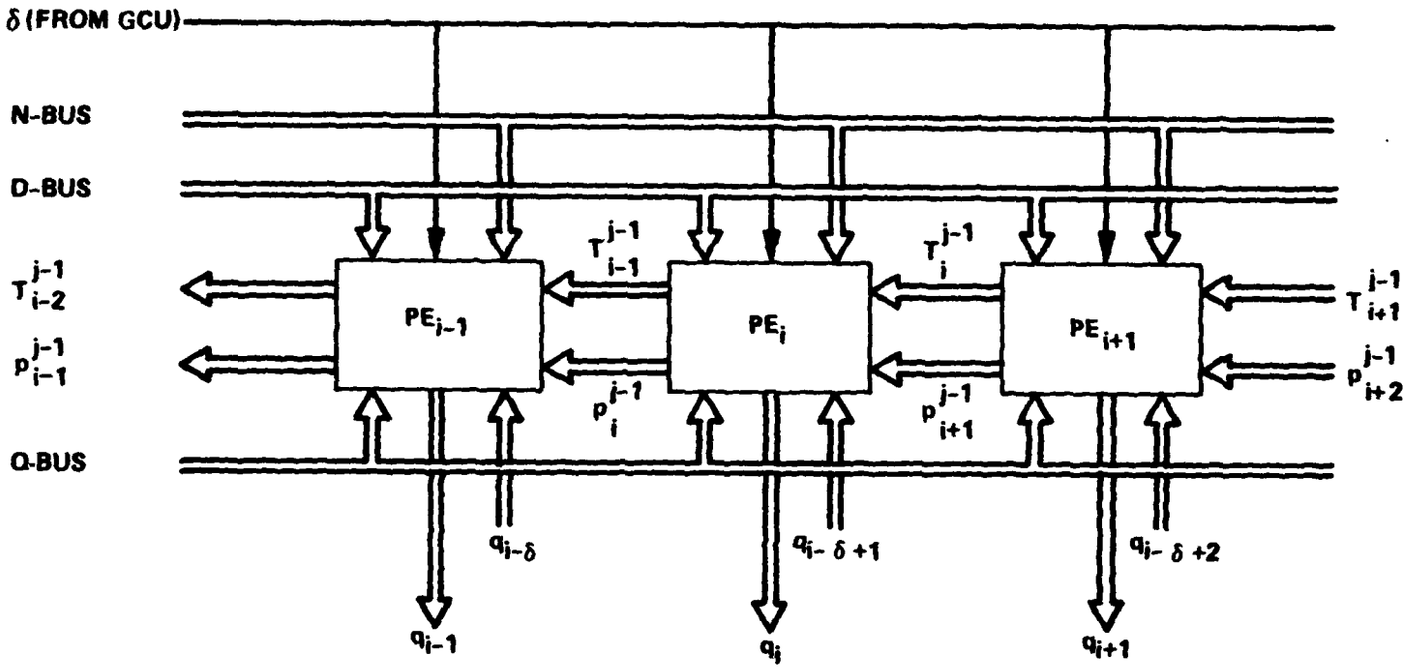


Figure A.2 – Interconnection Between Processing Elements

In order to eliminate carry propagation between the PEs, we assume that each digit of the partial remainder ($p_i^{(j)}$) in PE_i is represented by an interim partial remainder ($w_i^{(j)}$) and a transfer digit ($T_i^{(j)}$) such that:

$$p_i^{(j)}=w_i^{(j)}+T_i^{(j)} \tag{A.6}$$

The transfer function in Eq. (A.4) is obtained by a series of three transformations f_1 , f_2 and f_3 such that:

$$\begin{cases} f_1 : -q_{i+1-\delta}*d_{j+\delta}=rt_{i-1}^{p_1(j)}+w_i^{1(j)} \\ f_2 : -q_j*d_i=rt_{i-1}^{p_2(j)}+w_i^{2(j)} \end{cases} \tag{A.7}$$

The transfer digits from PE_i to PE_{i-1} are $t_{i-1}^{p_1(j)}$ and $t_{i-1}^{p_2(j)}$ resulting from transformations f_1 and f_2 respectively. Also there is a transfer digit out of the Multi-input Adder ($t_{i-1}^{A(j)}$). Therefore:

$$T_{i-1}^{(j)}=t_{i-1}^{p_1(j)}+t_{i-1}^{p_2(j)}+t_{i-1}^{A(j)} \tag{A.8}$$

substituting (A.6), (A.7) and (A.8) in (A.4) we get:

$$f_3 : \begin{cases} w_i^{(j)}=w_{i+1}^{(j-1)}+T_{i+1}^{(j-1)}+w_i^{2(j)}+w_i^{1(j)} \\ \qquad +n_{j+\delta}[i=\delta]-rt_{i-1}^{A(j)} \\ T_i^{(j)}=t_i^{p_1(j)}+t_i^{p_2(j)}+t_i^{A(j)} \\ p_i^{(j)}=w_i^{(j)}+T_i^{(j)}=w_i^{(j)}+t_i^{p_1(j)}+t_i^{p_2(j)}+t_i^{A(j)} \end{cases} \tag{A.9}$$

A block diagram of transformations f_1 , f_2 and f_3 is shown in Figure (A.3).

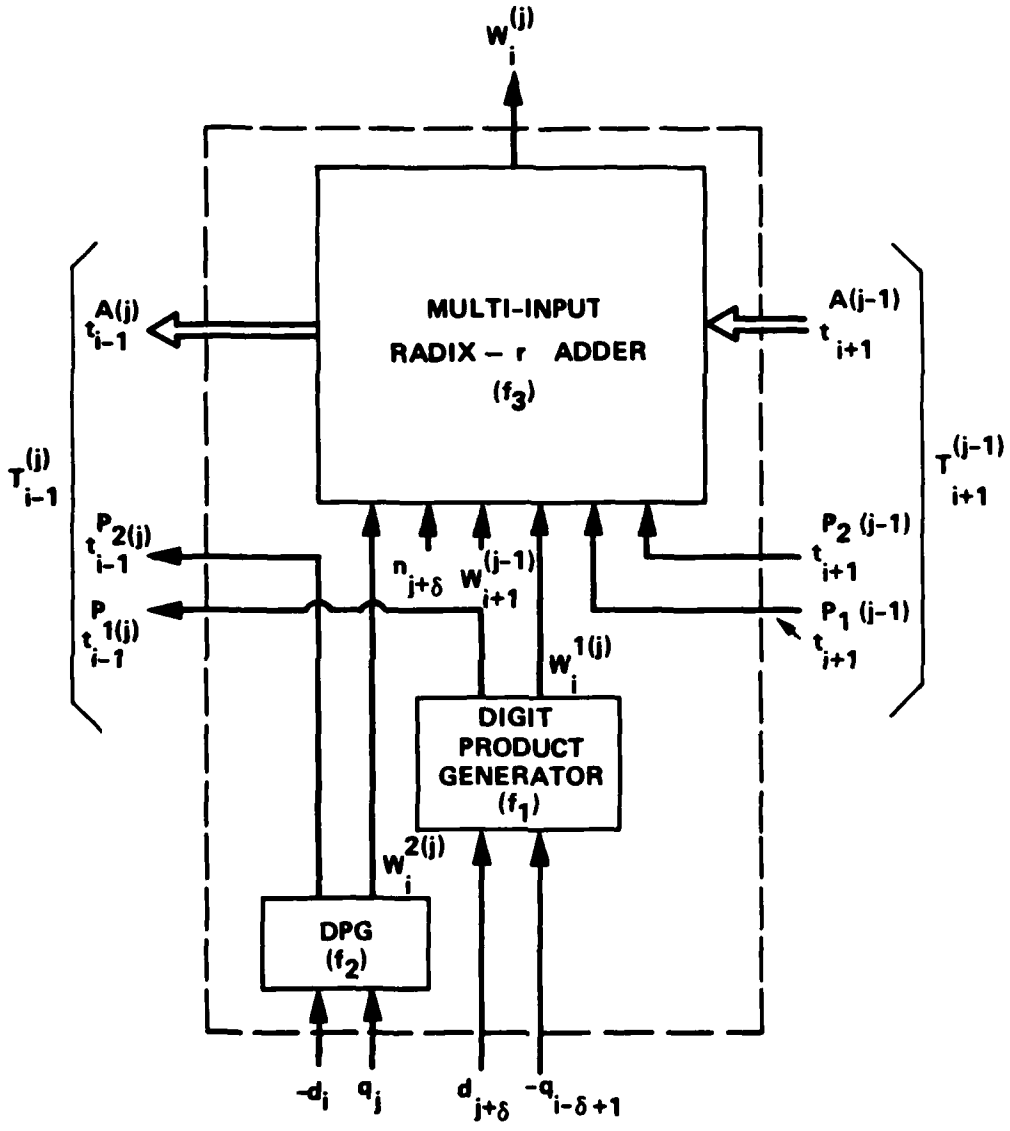


Figure A.3 – Functional Representation of the Digit Algorithm for On-Line Division

Transformation f_3 essentially requires a radix-r multi-input adder which forms the sum of the digits of both signs. This adder is implemented as a k-stage ($r=2^k$) linear cascade of a radix-2 multi-input adder where each input of a radix-2 adder can assume three values $\{\bar{1},0,1\}$. The organization of this adder is shown in Figure (A.4).

The products q_j*d_i and $q_{i+1-\delta}*d_{j+\delta}$ are generated by two separate product matrix generators which consist of a k*k square array of redundant binary product cells. Each cell performs the product of two redundant binary digits $q^*_{j_1}$ and $d^*_{i_m}$ and its output product digit is also in the digit set $\{\bar{1},0,1\}$. Figure (A.5) shows the operation of the digit product generators f_1 and f_2 (k=4).

Therefore, transformation f_3 requires k MIRBAs (Multi-Input Redundant Binary Adder), each capable of summing 2(k+1) redundant binary inputs, as well as the 'Transfer' from the adjacent MIRBA position [GOY 76]. Figure (A.6) schematically shows the implementation of f_3 for radix 16, that is, k=4.

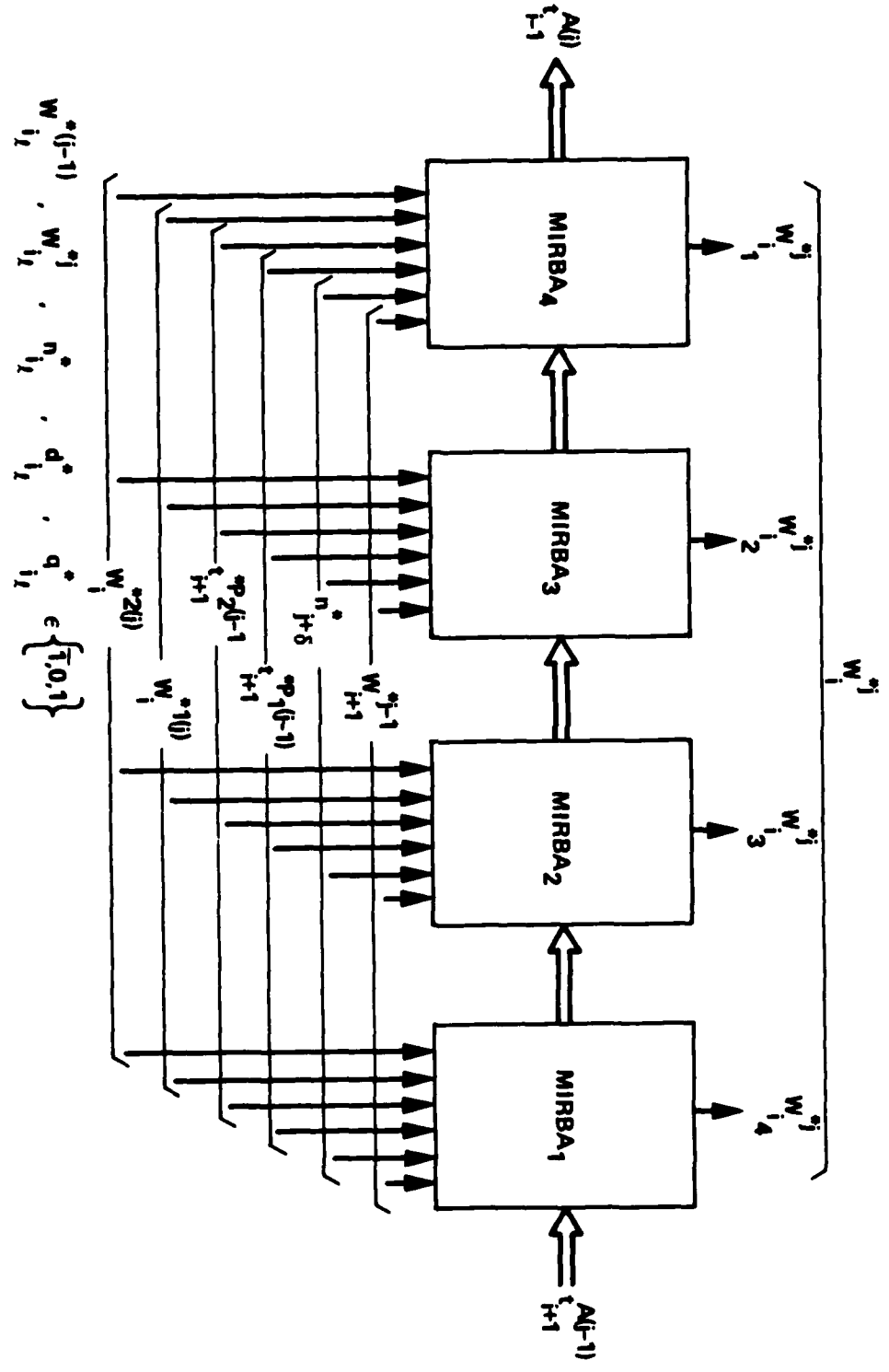


Figure A.4 – Functional Representation of Transformation f_3

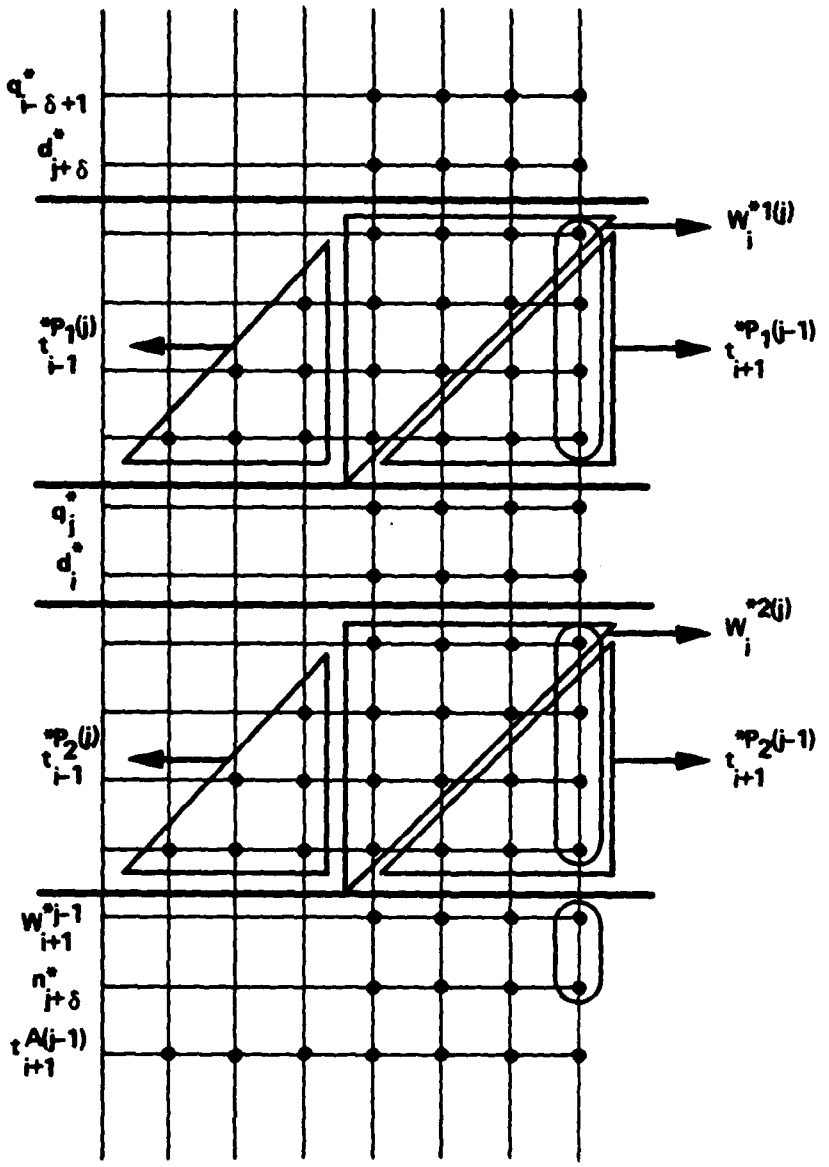


Figure A.5 – Illustration of Adjacent Overlapping Product Matrices

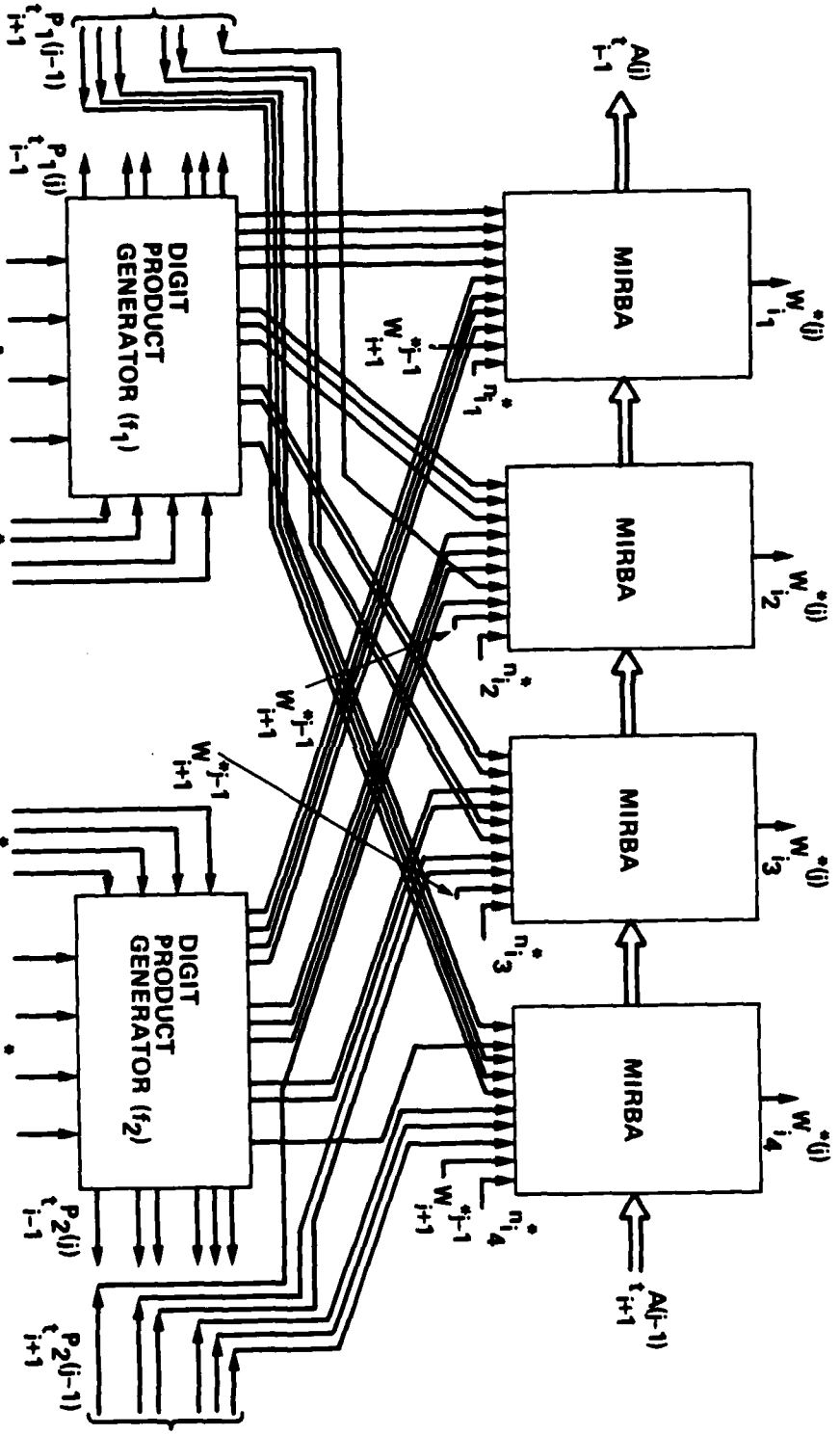


Figure A.6 – Illustration of The Implementation of The Digit Algorithm, Using Redundant Binary Product Matrix Generator (Radix = 16).

A.1 Design of a MIRBA

MIRBA is a limited carry/borrow propagation adder which accepts several redundant binary inputs (digit set [illegible]) and produces one redundant binary output (with appropriate adder Transfers for more significant adjacent adder stages).

Using Rohatsch's technique [ROH 67], a 10 input MIRBA can be realized with four simple transformations. Figure (A.7) shows one such four level (each level indicated by a box) adder which is applicable for $k \leq 6$.

Another way of implementing MIRBA's is the Log-sum tree technique. In this scheme each MIRBA can be implemented by a log-sum tree structure of two input redundant binary adders (Borovec Unit [BOR 68]). For a $2(k+1)$ input MIRBA, the tree structure has L levels of Borovec Units (BU) such that:

$$L=\lceil \log_2 2(k+1) \rceil \qquad (A.10)$$

and the number of BU's required is $(2k+1)$. Figure (A.8) shows the log-sum tree structure for a 10 input MIRBA.

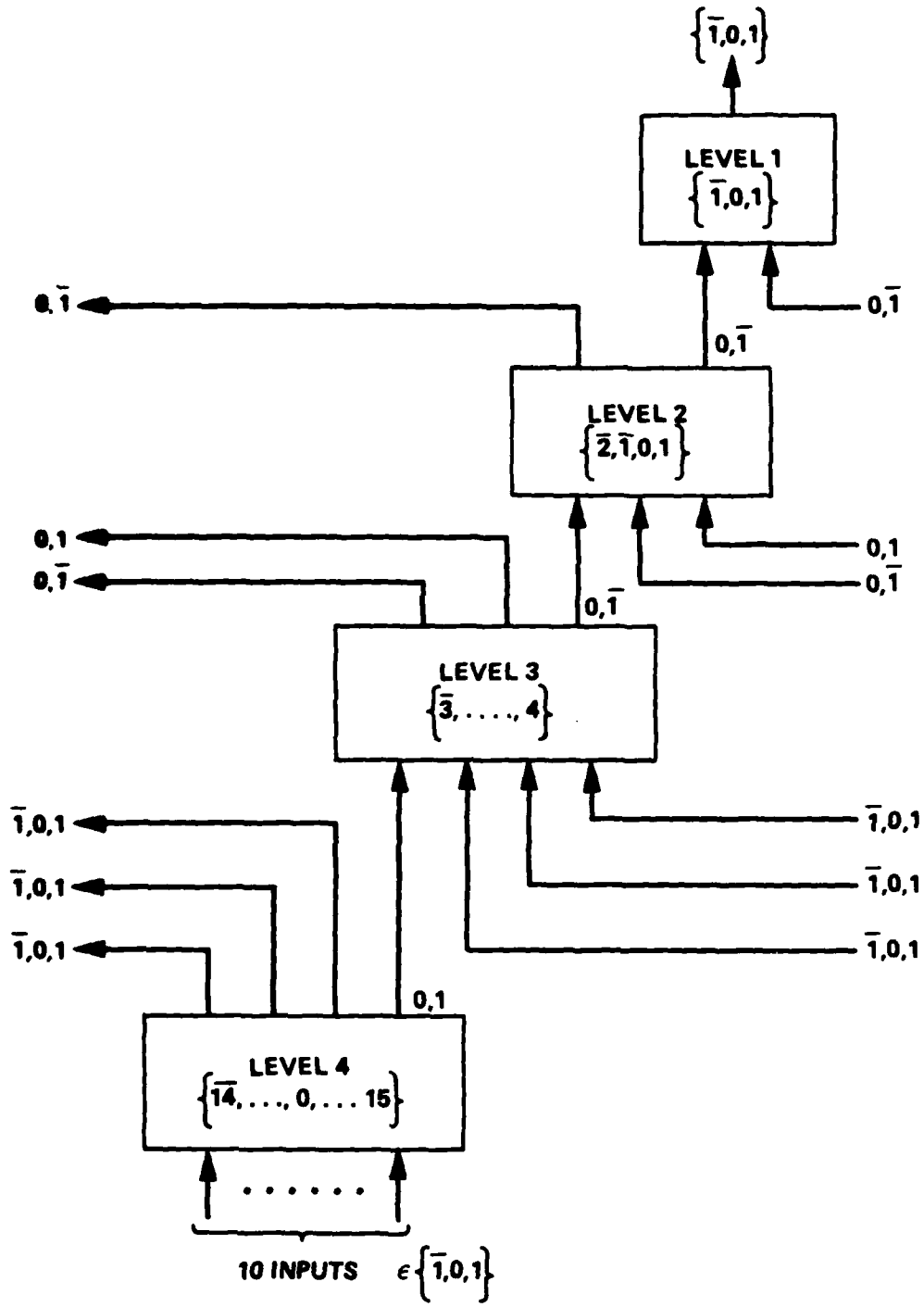


Figure A.7 – Illustration of the Algebraic Design of a MIRBA Using Simple Transformations

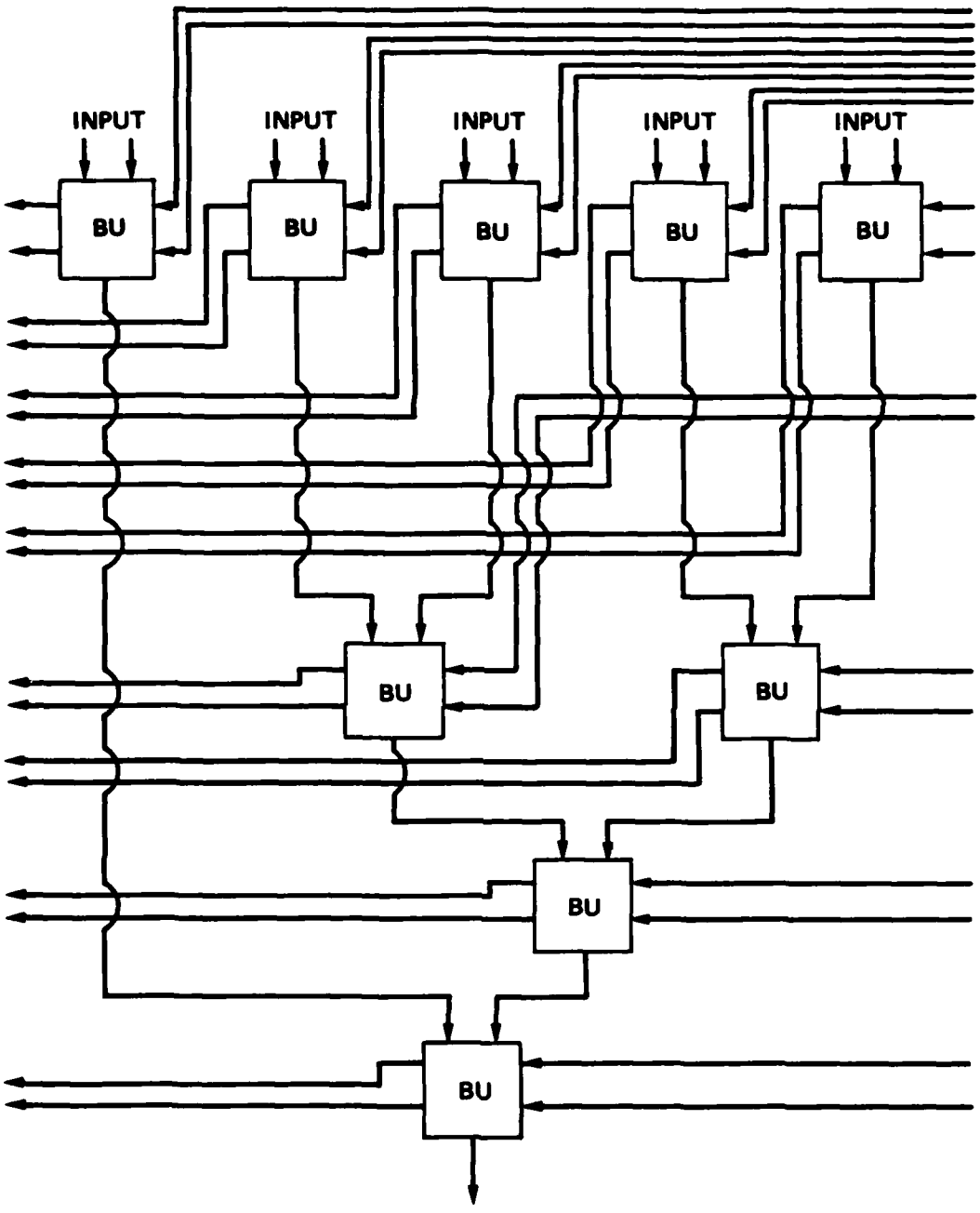


Figure A.8 – Illustration of Log-Sum Tree Structure for a MIRBA Using Borovec Units Only. (K=4)

A.2 Logic Design of The Processing Element

The major components of the PE are the Register File for the storage of active operands, The Digit Processing Logic (DPL) which is essentially a large combinational logic circuit and Local Control Unit (LCU) which supplies the control signals in proper order to condition the combinational DPL. Figure (A.9) shows the schematic block diagram of a Processing Element. The register file comprises a set of digit-wide registers which are used to hold the operand digits and the result digits.

The DPL operates on the operand digits stored in the register file of the PE and the informations received from its right neighboring PEs. It also generates Transfer information for its left neighbor PE. The LCU issues the timing control signals to the processing logic for sequencing the various steps of the digit algorithm.

The register file is a set of registers that are used to hold the operands and result digits. Each PE retains one digit of each of the active operands. Each register is (k+1) bits long to hold the k-magnitude bits and one sign bit of one sign and magnitude encoded radix-2^k digit.

There must be at least seven registers in a PE. One for the dividend, one for divisor, one for quotient digit and one for interim partial remainder ($w_i^{(j)}$). Three other regis-

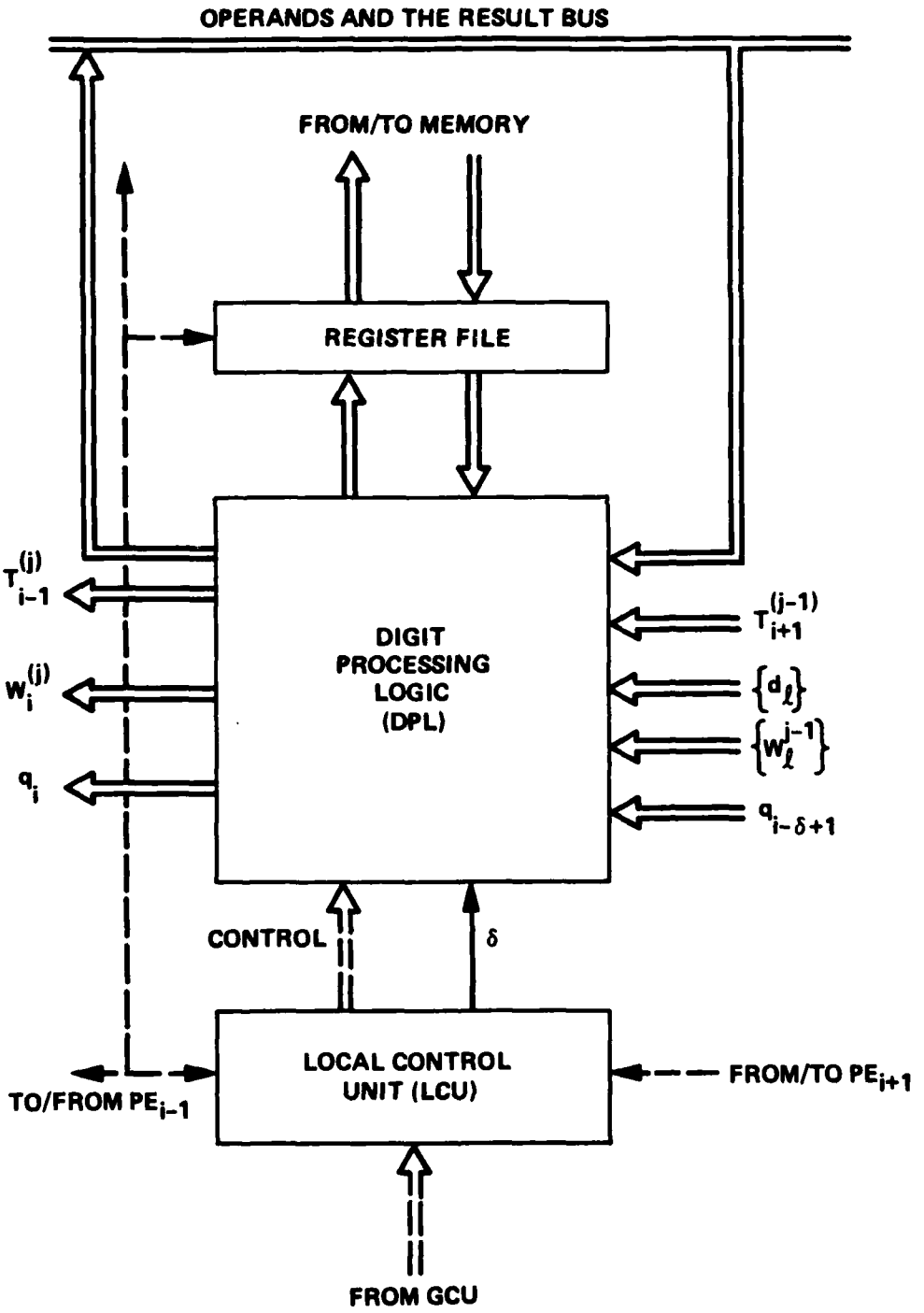


Figure A.9 – Block Diagram of a Processing Element.

ters are used to hold the transfer functions ($T_i^{(j)}$) coming from PE_{i+1}. In the next step of the computation (j+1) these functions are gated to PE_{i-1} along with $w_i^{(j)}$. They constitute the operands of PE_{i-1} in step (j+1).

There are other registers in a PE which are used to hold the intermediate results. These registers are located in DPL and will be shown later.

The registers in the register file are loaded from a buffer register, IBR whose contents are determined by the internal Register Input Bus Selector, SRIB in the Digit Processing Logic. Similarly, the contents of the registers are inputed to the DPL either directly or through an Output Bus Selector SROB, also in DPL.

A.3 Block Diagram Description of DPL

Figure A.10 shows the data flow structure of the Digit Processing Logic (DPL) in a block diagram form. It consists of three major components- the Digit Product Generator, DPG, a radix-2^k multi-input adder MIAD, and a Digit Sum Encoder, DSE. DSE converts the redundant binary sum output of adder MIAD to the Sign and Magnitude format for local storage in the Register File, or transfer out of the PE.

As shown in Figure (A.10) are input and output ports designated as TIP_i , RIP_i and TOP_i, ROP_i, respectively. The

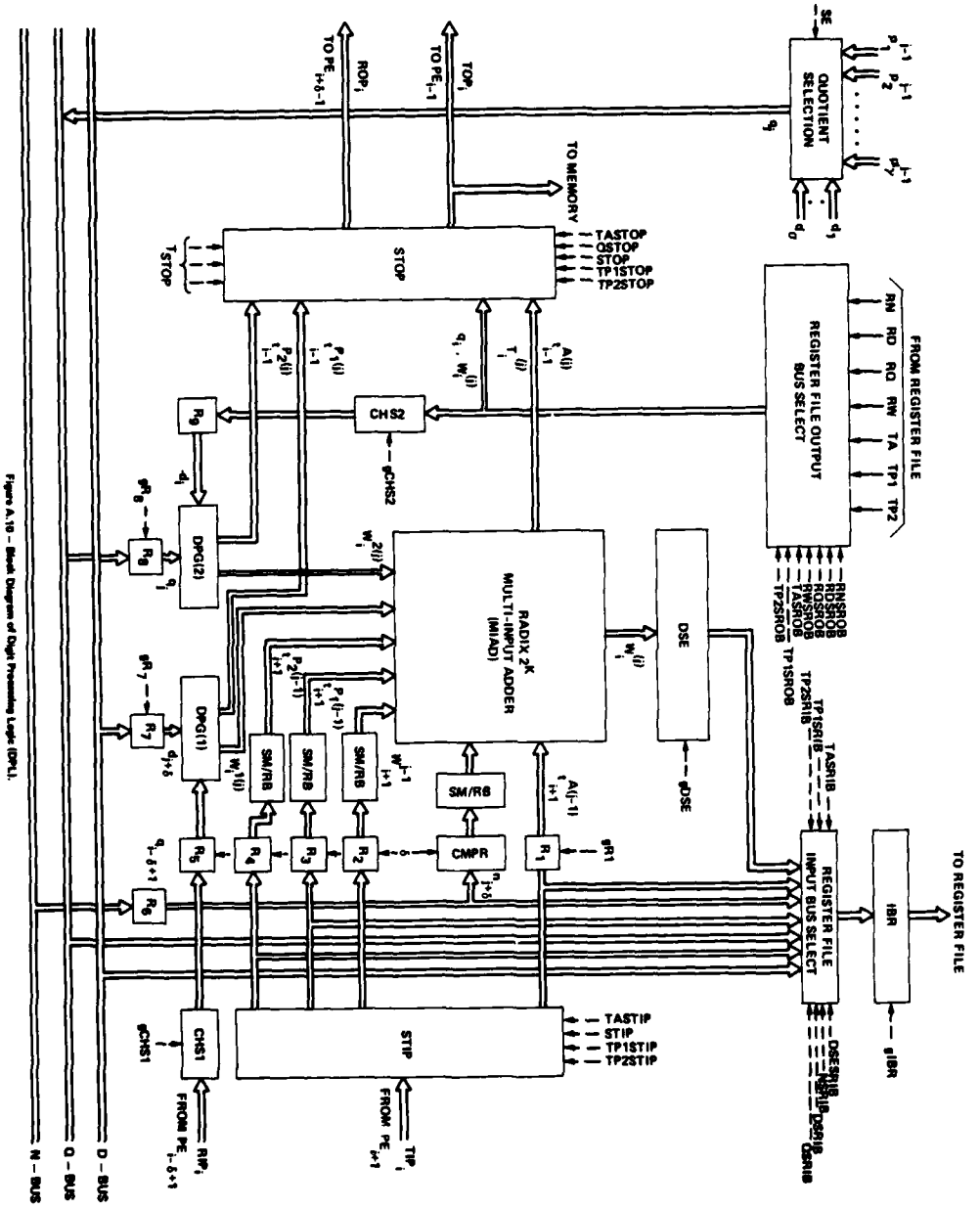


Figure A.10 – Block Diagram of Digit Processing Logic (DPL).

input port TIP_i carry the 'Transfer' (carry or borrow) from adjacent MIAD and the contents of some register in the Register File of the adjacent PE_{i+1}. RIP_i carry the quotient digit from $PE_{i+1-\delta}$. The output ports TOP_i and ROP_i carry similar information for PE_{i-1} and $PE_{i+1-\delta}$ respectively.

A.4 Logic Design of a Radix 2^k Multi-Input Adder (MIAD)

In general, a radix-2^k multi-input adder consists of a linear cascade of k MIRBAs. A $2(k+1)$ input MIRBA is implemented as a tree structure of BUs (see Fig. (A.8)). Each MIRBA requires $2k+1$ BUs and are arranged in $L=\lceil \log_2 2(k+1) \rceil$ levels. Therefore:

$$G_{MIAD}=k(2k+1)G_{BU} \tag{A.11}$$

$$t_{MIAD}=L*\delta_{BU} \tag{A.12}$$

G_{MIAD} =Number of Gates Required for One MIAD

t_{MIAD} = Delay of One MIAD

G_{BU} = Number of Gates Required for One BU

δ_{BU} = Delay of One BU

For a 2(k+1) input adder, the number of pins required for the input and output adder transfers $t_{i+1}^{A(j-1)}$ and $t_{i-1}^{A(j)}$ are $2(2k+1)$ each (see Figure A.8).

A.5 Logic Design of DPG

The Digit Product Generator forms the product array of two signed radix-2^k digits. It accepts the two digits encoded in Sign and Magnitude format and outputs the product array in redundant binary. The logical design of DPG is shown in Figure (A.11).

The number of gates required for each DPG is [GOY 76]:

k^2 AND GATES

1 XOR

$$\text{SM/RB} \begin{cases} k^2 \text{ XOR or } 2k^2 \text{ AND} & \text{for } LVE_1 \\ k^2 \text{ AND} & \text{for } LVE_2 \\ \text{NONE} & \text{for } LVE_3 \end{cases} \qquad \text{(A.13)}$$

The pins contributed by DPGs to the pin complexity of DPL are those pins which are required for $t_{i-1}^{p_1(j)}$, $t_{i-1}^{p_2(j)}$, $t_{i+1}^{p_1(j-1)}$ and $t_{i+1}^{p_2(j-1)}$.

$$\text{No. of pins for a transfer signal} = 1 + \frac{k(k-1)}{2} \qquad \text{(A.14)}$$

A.6 Logic Design of Digit Sum Encoder

The Digit Sum Encoder (DSE) transforms the redundant binary sum output of the radix-2^k adder into an algebraically equivalent radix-2^k sum digit in Sign and Magnitude format for either local storage in the Processing Element or

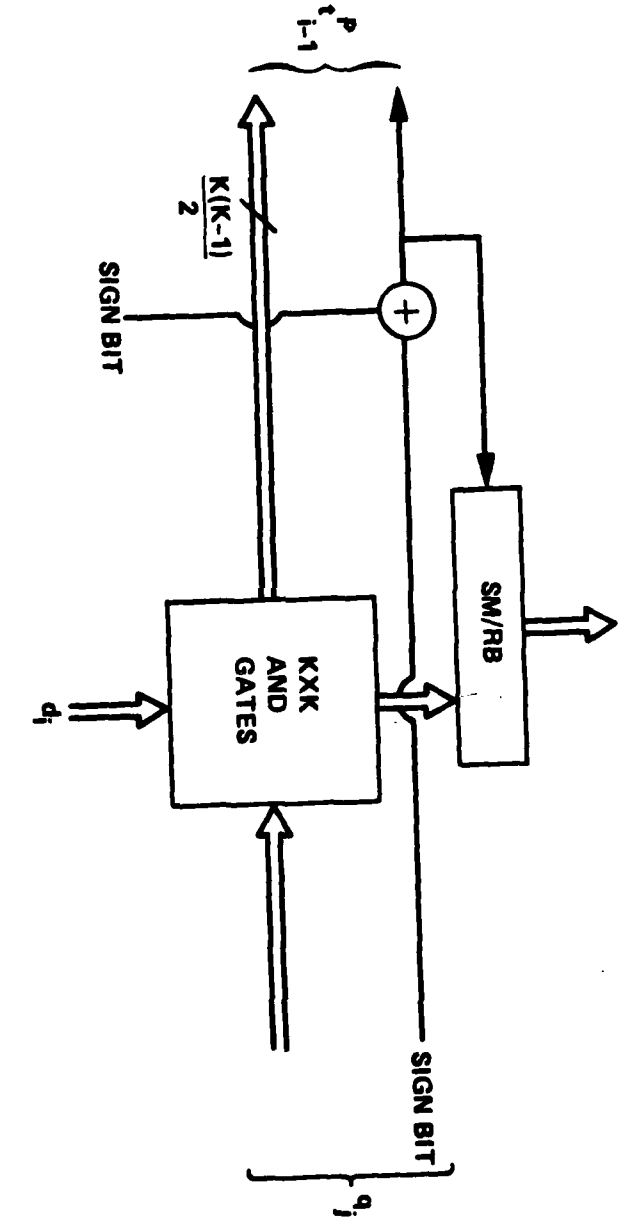


Figure A.11 – Logical Design of The DPG

transmission out of the PE. Total number of gates, G_{DSE} required by DSE logic has been found to be [GOY 76]:

$$G_{DSE} \tilde{=} \begin{cases} 16k \text{ for } LVE_2 \text{ and } LVE_3 \\ 26k \text{ for } LVE_1 \end{cases} \qquad (A.15)$$

A.7 Logic Design of Selectors SRIB, SROB, STOP and STIP

The selector SRIB is a seven input multiplexor. It constantly examines the data on D, Q and N Busses. If the data on any of these busses belong to PE_i, it writes this data in the corresponding registers in the register file. It also gates the output of DSE ($w_i^{(j)}$) to Register RW in the Register File. The transfer function $T_i^{(j)}$ ($=t_i^{A(j)}, t_i^{p_1(j)}, t_i^{p_2(j)}$) which should be sent to PE_{i-1} in (j+1)-th step is gated through this selector to Register File for temporary storage. The width of the selector is obtained by the following equation:

$$b=MAX\left[k+1, P_{t_i^{A(j)}}, P_{t_i^{p_1(j)}}, P_{t_i^{p_2(j)}}\right]$$

$P_{t_i^{A(j)}}$ = Pin Count of $t_i^{A(j)} = 2(2k+1)$

$P_{t_i^{p_1(j)}}$ = Pin Count of $t_i^{p_1(j)} = 1 + \frac{k(k-1)}{2}$

$P_{t_i^{p_2(j)}}$ = Pin Count of $t_i^{p_2(j)} = 1 + \frac{k(k-1)}{2}$

Therefore:

$$b = 2(2k+1) \qquad (A.16)$$

The logic design of SRIB is similar to that shown in [GOY 76] and the number of gates required is:

$$G_{SRIB} \tilde{} b+b+2(1+\frac{k(k-1)}{2})+4(k+1) = k^2+11k+10 \qquad (A.17)$$

The selector SROB selects the contents of one of the registers of the Register File on to the Register File Output Bus (ROB). The gates required for this network are dependent on the number of registers in the Register File and the bit width of the registers. There are seven registers in the Register File. For radix-2^k , four of them are (k+1) bits wide, one is 2(2k+1) bits wide and the other two are $(1+\frac{k(k-1)}{2})$ bits wide. Therefore, the gate requirements of SROB are exactly same as that of SRIB, that is:

$$G_{SROB} = k^2+11k+10 \qquad (A.18)$$

The width of selector STOP is equal to the width of output port TOP_i. The width of TOP_i is determined by the maximum length of "Adder Transfers". Therefore, the width of TOP_i is given by Eq. (A.16). Logic implementation of STOP is shown in Figure (A.12). From the given design we conclude that:

$$G_{STOP}=3b+2(1+\frac{k(k-1)}{2})=k^2+11k+8 \qquad (A.19)$$

The selector STIP is actually a four output demultiplexor. The width of STIP is exactly the same as that of STOP and is therefore equal to b. The logic implementation of STIP is simple and the number of gates required for this element is:

$$G_{STIP}=b+k+1+2(1+\frac{k(k-1)}{2})=k^2+4k+5 \qquad (A.20)$$

Figure (A.13) shows the logic implementation of this selector.

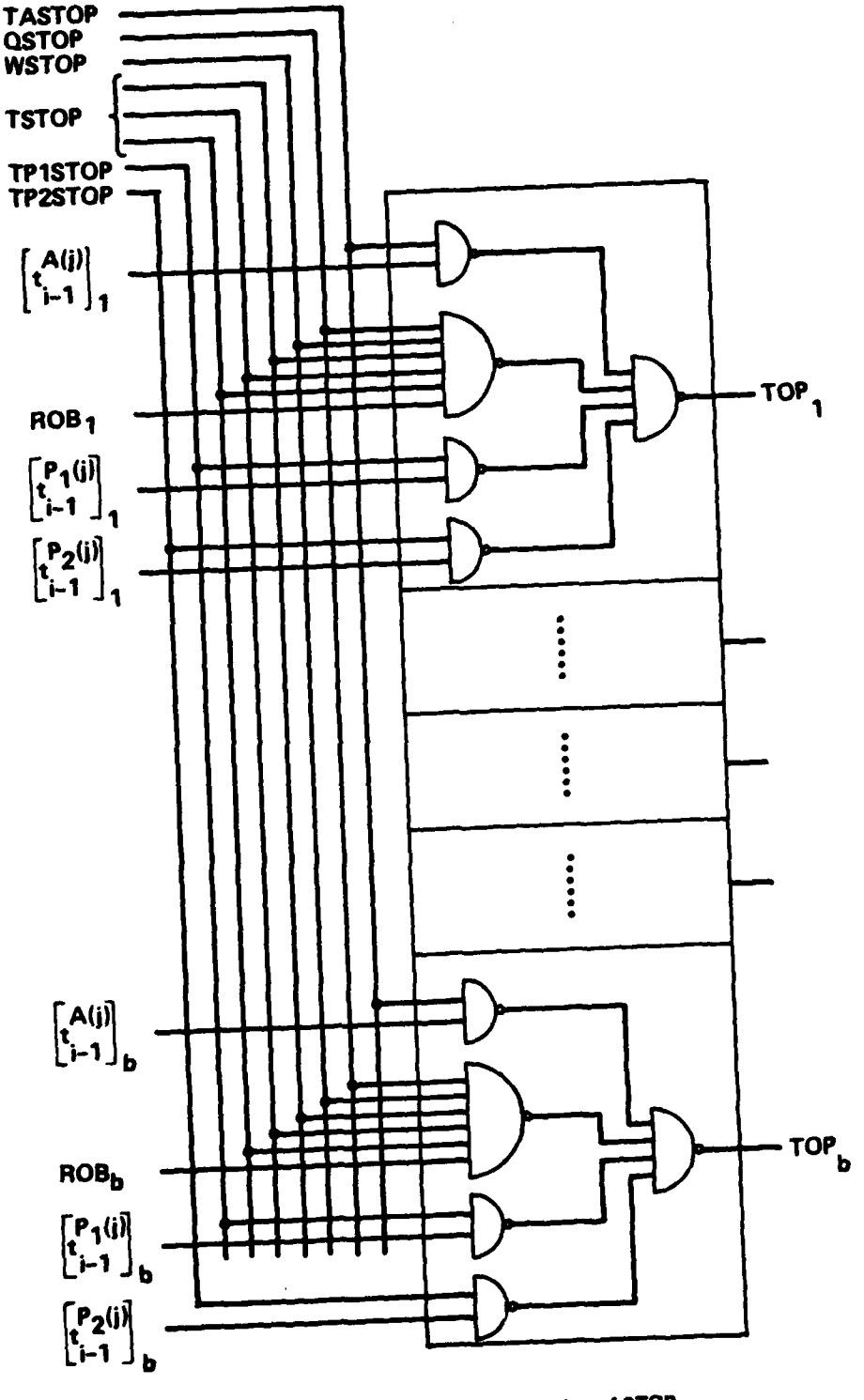


Figure A.12 – Logic Implementation of STOP

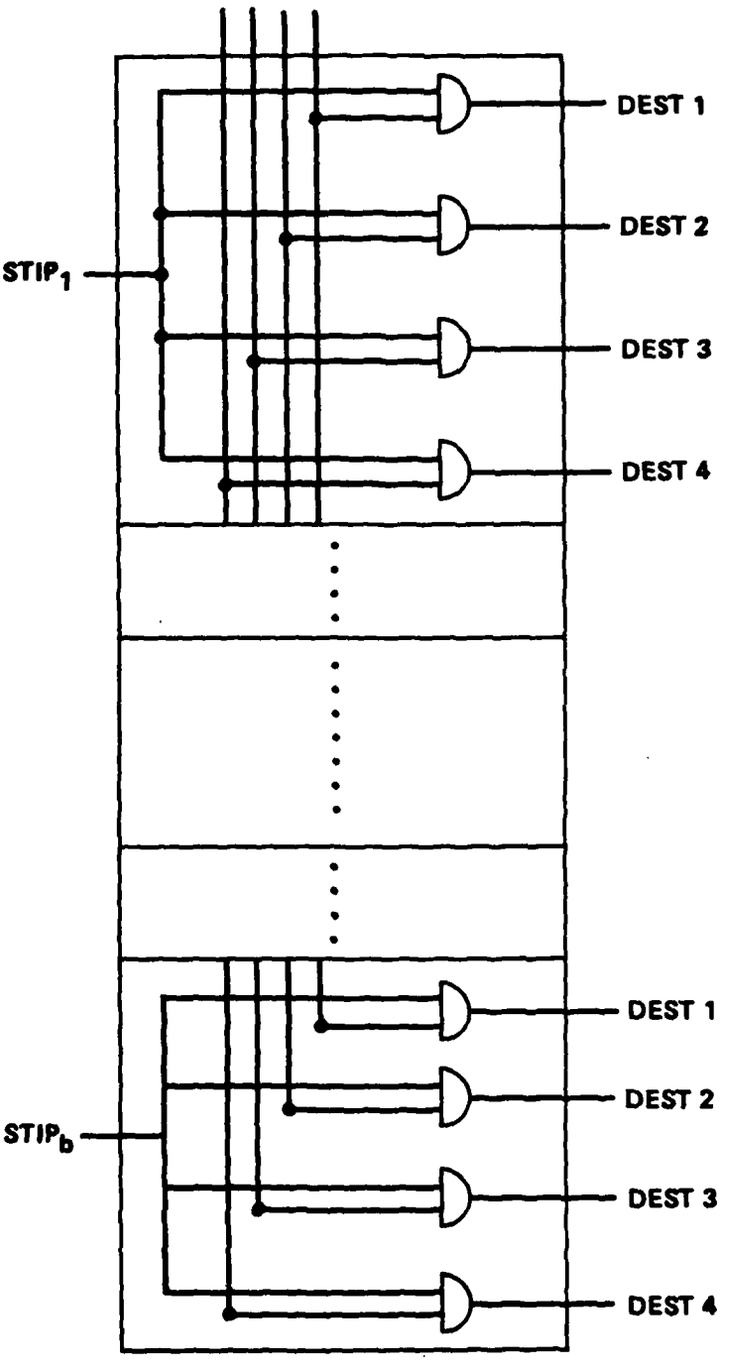


Figure A.13 – Logic Implementation of STIP

A.8 Storage Buffer Registers of DPL

DPL has ten buffer registers, R_1 through R_9 and IBR. The width of each of these registers has been indicated in Table (A.1).

Buffer Reg	Width bits
R_1	2(2k+1)
R_2	k+1
R_3	1+k(k-1)/2
R_4	1+k(k-1)/2
R_5	k+1
R_6	k+1
R_7	k+1
R_8	k+1
R_9	k+1
IBR	2(2k+1)

Table (A.1)- Width of The Registers in DPL

A.9 Design of SM/RB, CHS1 and CHS2 Blocks

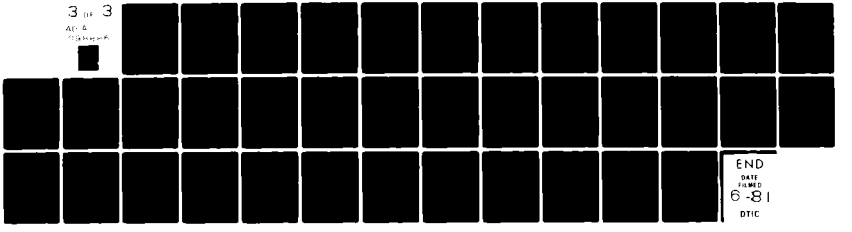
SM/RB block encodes the input which is represented in Sign and Magnitude format to redundant binary representation. There are nine distinct ways that we can encode a sign and magnitude number. The simplest one is the encoding that assigns the sign of the number to all the bits. Adopting this simple encoding, there is no gate requirement for SM/RB block. Therefore:

$$G_{SM/RB}=0 \tag{A.21}$$

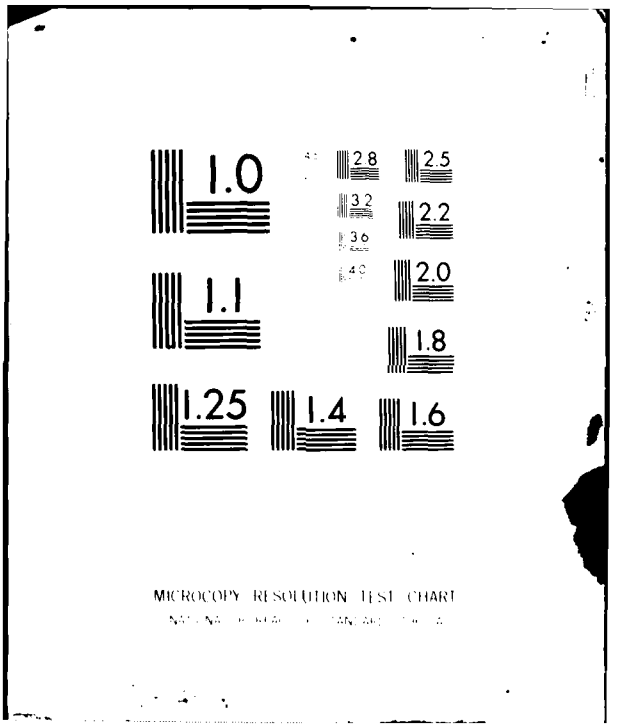
CHS1 and CHS2 are sign changers and since their inputs are in Sign and Magnitude format, they can be implemented by a single inverter gate. Therefore:

$$G_{CHS1}=G_{CHS2}=1 \tag{A.22}$$

AD-A098 886 CALIFORNIA UNIV LOS ANGELES DEPT OF COMPUTER SCIENCE F/G 9/2
ERROR-CODED ALGORITHMS FOR ON-LINE ARITHMETIC.(U)
FEB 81 A GORJI-SINAKI N00014-79-C-0866
UNCLASSIFIED UCLA-ENG-8197 NL



END
DATE FILMED
6-81
DTIC


1.0
2.8
2.5
3.2
2.2
3.6
4.0
2.0
1.1
1.8
1.25
1.4
1.6
MICROCOPY RESOLUTION TEST CHART

A.10 Design of The Quotient Selection Unit

The selection of the quotient digits is done by the most significant Processing Element (PE_1). The quotient digit selector inside PE_1 is a table look-up device which implements the SELECT function (see Algorithm MAIN DIVIDE). It examines γ most significant digits of rP_{j-1} and σ most significant digits of D_{j-1}, in order to select the appropriate quotient digit, q_j.

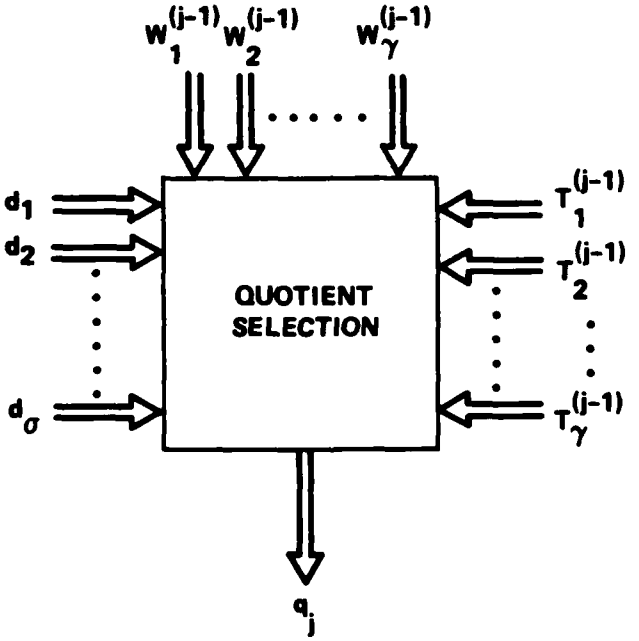


Figure (A.14)- Quotient Selection Unit

According to Eq. (A.9) :

$$P_{j-1} = \sum_{i=1}^{m} p_i^{(j-1)} * r^{-i} = \sum_{i=1}^{m} [w_i^{(j-1)} + T_i^{(j-1)}] r^{-i} \qquad (A.23)$$

Therefore, truncated version of P_{j-1} (i.e., $\hat{P}_{j-1}$) is:

$$\hat{P}_{j-1} = \sum_{i=1}^{y} [w_i^{(j-1)} + T_i^{(j-1)}] r^{-i} \qquad (A.24)$$

This means T_i's and w_i's can be used as the address lines of an ROM device implementing the SELECT function. It is not difficult to see that even for small radices the number of input lines to the device will be prohibitive [IRW 77]. Two techniques to avoid this dilemma are available: 1) Use a PLA, or 2) Perform Carry Propagation on the most significant portion of $\hat{P}_{j-1}$ to reduce the number of lines required. Irwin shows that the number of input line will be reduced by up to 44% if this technique is used [IRW 77]. In estimating the cost of the Processing Elements we have ignored the cost of the Selection Block. Because, it effectively appears in only one PE (PE_1). The time required by the selection process has been estimated to be of the order of 4-5 gate delays [ERC 75, IRW 77]. In the delay analysis of the division unit we assume

$$t_{select} = t_s = 4\delta_g$$

A.11 Gate Complexity of Digit Processing Logic

The total number of gates we require for the implementation of DPL is the sum of all the gates we require for each of its components. From Equation (A.11) we have:

$$G_{MIAD}=k(2k+1)G_{BU}$$

Each Borovec Unit (BU) requires 26 gates [GOY 76]. Therefore, the total number of gates required for the Multi-input Adder is:

$$G_{MIAD}=26k(2k+1)$$

other components of DPL require the following number of gates:

$$G_{DPG(1)}=k^2 \text{ AND } +2 \text{ XOR } =k^2+8 \text{ GATES}$$

$$G_{DPG(2)}=k^2+8 \text{ GATES}$$

$$G_{DSE}=16k$$

$$G_{SRIB}=k^2+11k+10$$

$$G_{SROB}=k^2+11k+10$$

$$G_{STOP}=k^2+11k+8$$

$$G_{STIP}=k^2+4k+5$$

$$G_{SM/RB}=0$$

$$G_{CHS1} = G_{CHS2} = 1$$

Adding these together we get:

$$G_{DPL} = 58k^2 + 79k + 51 \qquad (A.25)$$

Table (A.2) shows the gate complexity of DPL.

A.12 Pin Complexity of DPL

The pins required for digit processing logic DPL is the sum of the pins necessary for input ports TIP_i, RIP_i and output ports TOP_i and ROP_i. The total number of pins, P_{DPL} necessary for logic implementation of DPL is equal to the sum of the pins required for input and output ports.

$$P_{DPL} = P_{TOP_i} + P_{ROP_i} + P_{TIP_i} + P_{RIP_i}$$

from (A.22) we have:

$$P_{TIP_i} = P_{TOP_i} = P_{t_{i+1}A(j-1)} = 2(2k+1) \qquad (A.26)$$

and since the information on RIP_i is a single digit then:

$$P_{ROP_i} = P_{RIP_i} = k+1 \qquad (A.27)$$

plugging (A.26) and (A.27) in the equation for P_{DPL} we get:

$$P_{DPL} = 10k + 6 \qquad (A.28)$$

r	K	G_{MIAD}	G_{DPG}	G_{DSE}	G_{SRIB}	G_{SROB}	G_{STOP}	G_{STIP}	G_{DPL}
4	2	260	12	32	36	36	34	17	441
8	3	546	17	48	52	52	50	26	810
16	4	936	24	64	70	70	68	37	1295
32	5	1430	33	80	90	90	88	50	1896
64	6	2028	44	96	112	112	110	65	2613
128	7	2730	57	112	136	136	134	82	3446
256	8	3536	72	128	162	162	160	101	4395

Table A.2 – Gate Complexity of DPL VS Radix for LVE_3 Encoding of a Redundant Binary Digit.

A.13 Overall Logic Complexity of a PE

The total number of gates, G_{PE}, required for the implementation of a PE is the sum of the gates required for the combinational logic of DPL, the gates required for the PE control logic and the gates required for the implementation storage registers in the PE. The storage registers in a PE comprise the registers in the Register File and buffer registers in DPL. Using Table (A.1) the number of gates needed for storage is:

$$G_{STO}=[6(k+1)+4(2k+1)+k(k-1)+2]G_D$$

$$=(k^2+13k+12)G_D$$

G_D is the number of gates required for the realization of a D type flip-flop. Assuming $G_D=6$ [TEX 69] we get:

$$G_{STO}=6k^2+78k+72 \qquad (A.29)$$

Ignoring the number of gates needed for PE control, the number of gates required for each PE is:

$$G_{PE}=G_{DPL}+G_{STO}$$

Substituting the values from Equations (A.25) and (A.29) we get:

$$G_{PE}=64k^2+157k+123 \qquad (A.30)$$

The pin requirements for each PE is the sum of pins required for DPL plus the number of pins needed for input and output busses (ignoring the pins required for control signal

from GCU). That is,

$$P_{PE}=P_{DPL}+P_{N-BUS}+P_{D-BUS}+P_{Q-BUS}$$

or

$$P_{PE}=13k+9 \qquad (A.31)$$

The pin and gate requirements of DPL and PE along with the gate requirement of other PE components have been shown in Table (A.3).

r	K	G_{MIAD}	G_{DPG}	G_{DSE}	G_{SRIB}	G_{SROB}	G_{STOP}	G_{STIP}	G_{DPL}	P_{DPL}	G_{STO}	G_{PE}	P_{PE}
2	1	78	9	16	22	22	20	10	188	16	156	344	22
4	2	260	12	32	36	36	34	17	441	26	252	693	35
8	3	546	17	48	52	52	50	26	810	36	360	1170	48
16	4	936	24	64	70	70	68	37	1295	46	480	1775	61
32	5	1430	33	80	90	90	88	50	1896	56	612	2508	74
64	6	2028	44	96	112	112	110	65	2613	66	756	3369	87
128	7	2730	57	112	136	136	134	82	3446	76	912	4358	100
256	8	3536	72	128	162	162	160	101	4395	86	1080	5475	113

Table A.3 – Gate and Pin Complexity of a Processing Element vs The RADIX (r)

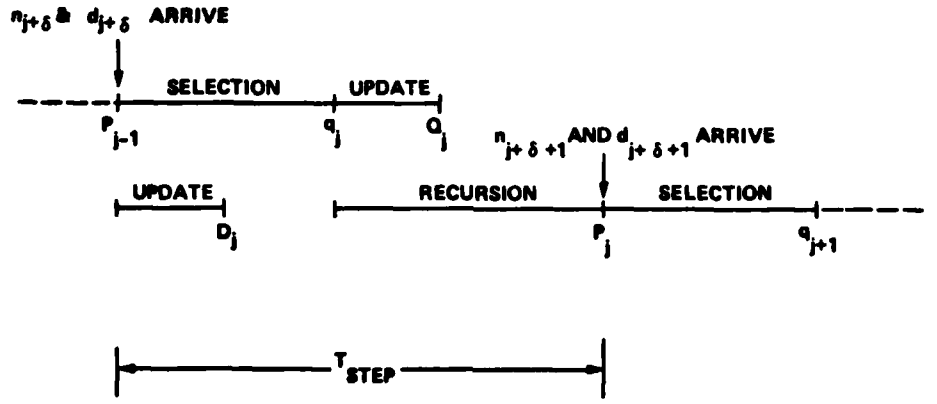
APPENDIX B

TIME (DELAY) CONSIDERATIONS OF AN ON-LINE DIVISION UNIT

Time required to compute a single quotient digit (k+1 bits) is composed of the following elements (see Algorithm MAIN DIVIDE).

1. Time to select a quotient digit (t_s)
2. Time to update Q_j and D_j registers (t_u)
3. Time to perform the basic recursion formula (t_R)

The following diagram indicates the relative position of these three delays with respect to one another.



Since usually t_s and t_R are greater than t_u, the total time for one step of the algorithm (T_{STEP}) is:

$$T_{STEP}=t_s+t_R \qquad (B.1)$$

Each step starts when the digits of the dividend ($n_{j+\delta}$) and divisor ($d_{j+\delta}$) appear on the input busses (N-BUS and D-BUS). At the beginning of each step selection of the quotient digit (q_j) is initiated by the quotient selection unit in the most significant Processing Element (PE_1). This selection is based on the truncated version of the previous partial remainder ($\hat{P}_{j-1}$) and divisor ($\hat{D}_{j-1}$).

PE_1 outputs q_j on the Q-BUS. After reception of this quotient digit and some other informations from its right neighbor, each PE starts processing of one digit of the next partial remainder (P_j). After certain amount of time (t_{PE}), next partial remainder will be available in a redundant format ($w_i^{(j)}$ and $T_i^{(j)}$). This process continues until required precision is obtained. We compute t_{PE} by measuring the time span between the setting on all registers (R_1 through R_9) in PE_i at step (j) and (j+1). Therefore (B.1) can be rewritten as:

$$T_{STEP}=t_s+t_{PE} \qquad (B.2)$$

Graph representation of t_s+t_{PE} is shown in Figure (B.1) [refer to block diagram of the MAIN Unit in Appendix-A]. Using this graph T_{STEP} is found to be:

$$T_{STEP}=2t_{SRIB}+t_{DSE}+t_{MIAD}+t_{SM/RB}+t_{STIP}+t_{STOP}$$

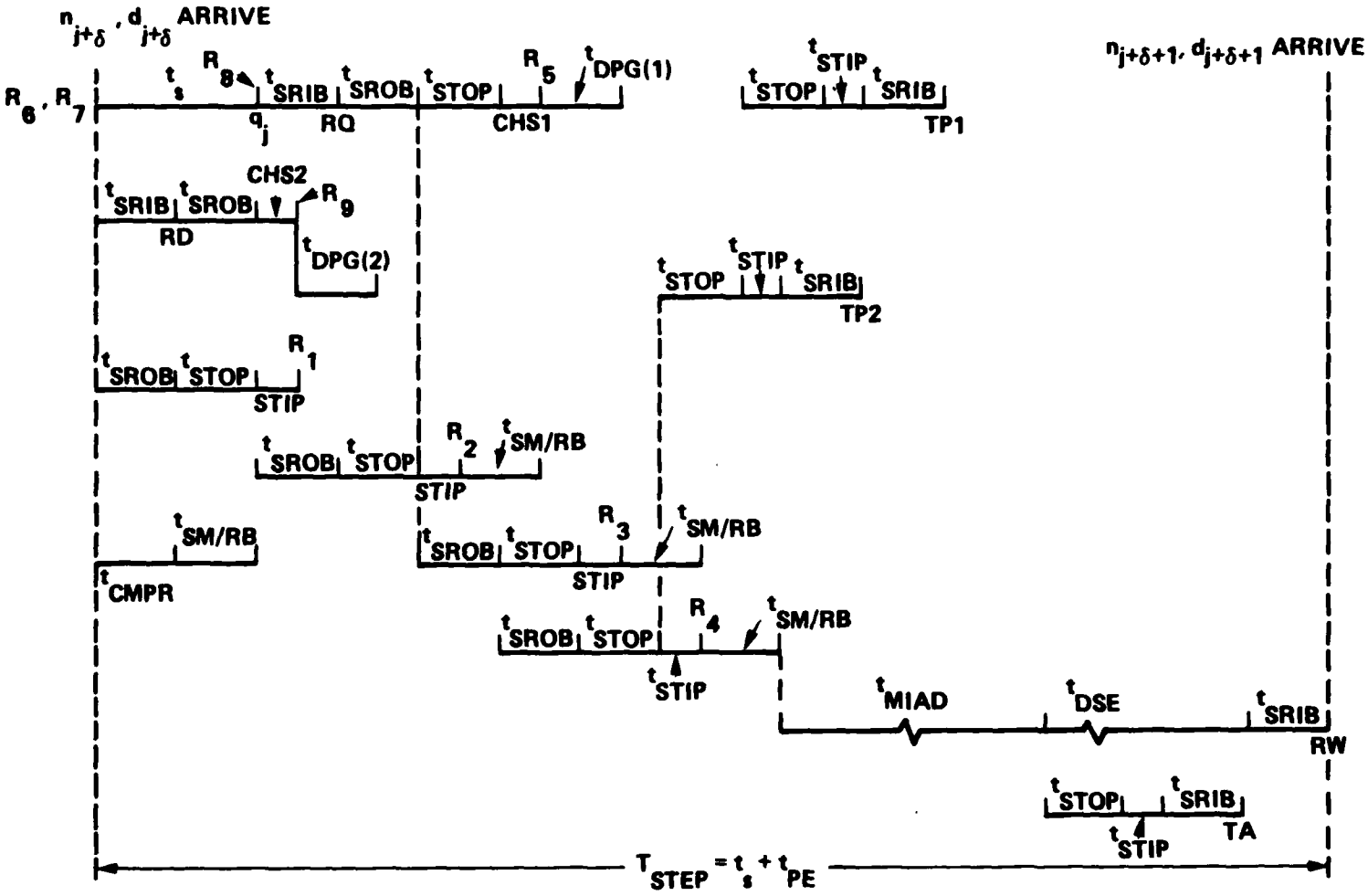


Figure B.1 – Graph Representation of T_{STEP}

$$+3t_{SROB}+t_s \qquad (B.3)$$

The components of T_{STEP} are as follows:

t_s:

The time required by the selection block has been estimated to be in the order of 4-5 gate delays [IRW 77]. Assume:

$$t_s=4\delta_g \qquad (B.4)$$

t_{SRIB}:

Logic design of Register File Input Bus Selector (SRIB) is given in Appendix-A. According to this design:

$$t_{SRIB}=2\delta_g \qquad (B.5)$$

t_{SROB}:

Referring to Appendix-A :

$$t_{SROB}=2\delta_g \qquad (B.6)$$

Also we have:

$$t_{STOP}=2\delta_g \qquad (B.7)$$

and

$$t_{CHS1}=\delta_g \qquad (B.8)$$

t_{DPG}:

Assuming LVE_3 (Logic Vector Encoding) for the operands [GOY 76], and according to what has been explained in the design of the Digit Product Generator we get:

$$t_{DPG}=t_{XOR}=2\delta_g \tag{B.9}$$

t_{MIAD}:

According to Eq. (A.12) in Appendix A :

$$t_{MIAD}=L*\delta_{BU} \tag{B.10}$$

such that: $L=\lceil \log_2 2(k+1) \rceil$ and δ_{BU} is the time required by one Borovec Unit [BOR 68]. Using LVE_3, δ_{BU} is obtained to be [GOY 76] :

$$\delta_{BU}=7\delta_g \tag{B.11}$$

Therefore:

$$t_{MIAD}=7\delta_g \lceil \log_2 2(k+1) \rceil \tag{B.12}$$

t_{DSE}:

From the design given in [GOY 76] t_{DSE} can be estimated approximately to be:

$$t_{DSE}\tilde{}5k\delta_g+3k\delta_g=8k\delta_g \tag{B.13}$$

t_{STIP}:

According to the design given in Appendix A :

$$t_{STIP}=\delta_g \tag{B.14}$$

Adding the components in Eq. (B.3) we get:

$$T_{STEP}=\left[8k+7\left\lceil\log_2(k+1)\right\rceil+24\right]\delta_g \tag{B.15}$$

Table (B.1) shows T_{STEP} and its components. From this table it can be deduced that contribution of "Digit Sum Encoder" (DSE) to the total step time dominates all other components for relatively large radices. But this unit can be eliminated if $w_i^{(j)}$ can be stored in redundant format. That is, RW and R_2 should be made to be double bank registers. Also STIP, SRIB, SROB and STOP blocks should be modified.

r	k	t_{SRIB} (δ_q)	t_{DSE} (δ_q)	t_{MIAD} (δ_q)	t_{STIP} (δ_q)	t_{STOP} (δ_q)	t_{SROB} (δ_q)	t_{STEP} (δ_q)
2	1	2	8	14	1	2	2	39
4	2	2	16	21	1	2	2	54
8	3	2	24	21	1	2	2	62
16	4	2	32	28	1	2	2	77
32	5	2	40	28	1	2	2	85
64	6	2	48	28	1	2	2	93
128	7	2	56	28	1	2	2	101
256	8	2	64	35	1	2	2	116

Table (B.1)- Time Required for One Step of the Division Process (T_{STEP}) and its Components

APPENDIX C

ON-LINE MULTIPLICATION

The problem of on-line multiplication has been addressed by Trivedi and Ercegovac [TRI 77] and by Irwin [IRW 77]. These two references deal with on-line multiplication when the operands are represented in a non-redundant number representation system.

The purpose of this appendix is to present a systematic method for derivation of on-line multiplication which is compatible with the method given for division [GOR 80]. The problem of redundant operand multiplication is addressed and it will be proved that the given upper bounds for the operands in the aforementioned references are pessimistic and the correct value will be derived.

Redundant Operands

Let the radix r representation of multiplicand, multiplier and the product be denoted by X, Y and R respectively such that:

$$X = \sum_{i=1}^{m} x_i r^{-i} \qquad (C.1)$$

$$Y=\sum_{i=1}^{m} y_i r^{-i} \tag{C.2}$$

$$R=\sum_{i=1}^{m} p_i r^{-i} \tag{C.3}$$

and R=X*Y to m digits of precision.

We assume x_i and y_i belong to the following redundant digit set:

$$x_i, y_i \in \{-\rho', \ldots, \bar{1}, 0, 1, \ldots, \rho'\} \quad (r-1) \geq \rho' \geq r/2 \tag{C.4}$$

p_i may belong to a different redundant digit set:

$$p_i \in \{-\rho, \ldots, \bar{1}, 0, 1, \ldots, \rho\} \quad (r-1) \geq \rho \geq r/2 \tag{C.5}$$

Redundancy coefficients of X, Y and R are defined as:

$$\begin{cases} k=\dfrac{\rho}{r-1} & (r-1) \geq \rho \geq r/2 \\ k'=\dfrac{\rho'}{r-1} & (r-1) \geq \rho' \geq r/2 \end{cases}$$

Assume that X and Y are bounded by a positive constant M such that:

$$-M \leq X, Y \leq M \tag{C.6}$$

M specifies the maximum and the minimum values that the operands can assume and is a function of r and ρ. This value will be derived later in this appendix.

The algorithm which produces the product of two redundant operands X and Y is called the Algorithm "MULT" and is shown below [TRI 77]:

Algorithm MULT

Step 1 [Initialization]:

$P_0=0, X_0=0, Y_0=0$

$R_0=0, p_0=0$

For j=1,2,....,m Do:

Step 2 [Input Digit]:

$X_j=X_{j-1}+x_j r^{-j}$

$Y_j=Y_{j-1}+y_j r^{-j}$

Step 3 [Basic Recursion]:

$$P_j=rP_{j-1}-rp_{j-1}+X_jY_j+Y_{j-1}x_j \qquad (C.7)$$

Step 4 [Selection]:

$p_j=\text{SELECT}(\hat{P}_j)$

$R_j=R_{j-1}+p_j r^{-j}$

Step 5 [End Do]

Proof of Convergence

Inserting different values of j into the basic recursion formula (Eq. C.7) we get:

$$j=1 \rightarrow P_1=X_1Y_1$$

$$j=2 \rightarrow P_2=rX_1Y_1+X_2Y_2+Y_1x_2-rp_1$$

$$=r^2X_2Y_2-rp_1$$

$$j=3 \rightarrow P_3=r^3X_3Y_3-(rp_1+p_2)$$

Continuing this procedure P_j is obtained as follows:

$$P_j=r^jX_jY_j-(r^{j-2}p_1+r^{j-3}p_2+\ldots+p_{j-1})$$

$$=r^jX_jY_j-r^jR_{j-1} \qquad (C.8)$$

when j=m

$$P_m=r^mX_mY_m-r^mR_{m-1} \qquad (C.9)$$

From (C.3) we have:

$$R=R_{m-1}+p_mr^{-m}$$

Inserting this in (C.9) we get:

$$P_m=r^mXY-r^m(R-p_mr^{-m})$$

or

$$R=XY-r^{-m}(P_m-p_m) \qquad (C.10)$$

By devising a product digit selection procedure, SELECT, in Step 4 of the Algorithm "MULT", such that:

$$|P_m - p_m| \leq k \qquad (C.11)$$

R=X*Y can be computed to m digits of precision. Note that the algorithm as it stands produces just the most significant half of the product. The least significant half of the product is available as the redundant output of the adder after iteration m+1, i.e.,

$$P_{m+1} = P_m - p_m \qquad (C.12)$$

By feeding these redundant adder digits directly into the recoding unit, the least significant half of the product can also be output in conventional form.

Range Restriction Analysis

Assume that the required SELECTION process in Step 4 of the Algorithm "MULT" is found and the graph of Figure (C.1) is obtained. This is a plot of partial product at step j versus partial product at step (j-1). This plot is designated as a P-P plot [IRW 77]. By analyzing such a plot, a product digit selection procedure can be specified for the given r and ρ. The notations used in this graph are similar to those used for division [GOR 80].

U_i - Upper bound for the region in which $p_{j-1}=i$
L_i - Lower bound for the region in which $p_{j-1}=i$
$(P_j)^i$ - The j-th partial product with $p_{j-1}=i$ chosen as the product digit.

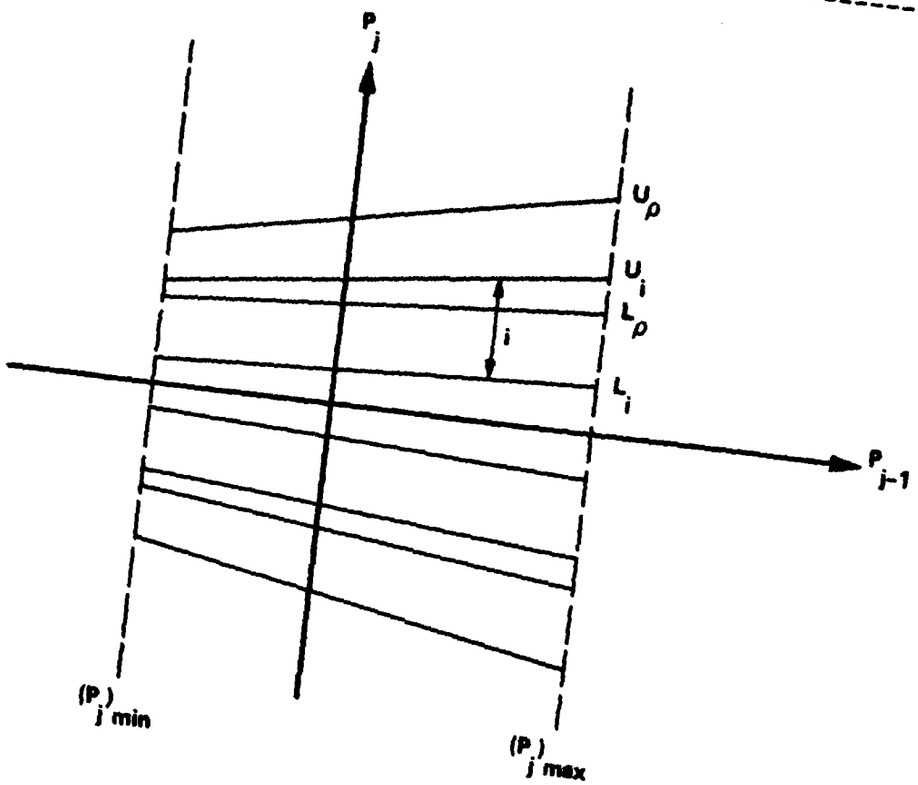


Figure (C.1)- A P-P plot

In order to derive the range restriction on P_j, the only thing we assume is the basic recursion formula (C.7) and the fact that P_j should be bounded by some constants C_1 and C_2.

$$\begin{cases} P_j = rP_{j-1} - rp_{j-1} + X_j Y_j + Y_{j-1} X_j \\ C_2 \geq P_j \geq C_1 \end{cases}$$

In step (j-1) assume $p_{j-1}=i$ is chosen. From the basic recursion formula (C.7) we are able to find the maximum value that P_j can assume.

$$(P_j)_{max}^{p_{j-1}=i} = rU_i - ri + 2M\rho' \qquad (C.13)$$

$$(P_j)_{min}^{p_{j-1}=i} = rL_i - ri - 2M\rho' \qquad (C.14)$$

When $p_{j-1}=\rho$ from (C.13) we get:

$$(P_j)_{max}^{p_{j-1}=\rho} = rU_\rho - r\rho + 2M\rho'$$

In order for P_j to be bounded, this value should be equal to U_ρ , therefore:

$$rU_\rho - r\rho + 2M\rho' = U_\rho$$

or

$$U_\rho = rk - 2Mk' \qquad (C.15)$$

Similarly for the lower bound:

$$(P_j)_{min}^{p_{j-1}=\rho} = L_{-\rho}$$

This results in:

$$L_{-\rho} = -rk + 2Mk' \qquad (C.16)$$

from (C.15) and (C.16) we get:

$$rk - 2Mk' \geq P_j \geq -rk + 2Mk' \qquad (C.17)$$

Selection Region

Selection regions can be obtained by the help of Equations (C.13), (C.14), (C.15) and (C.16) as follows:

$$(P_j)_{max}^{p_{j-1}=i} \leq U_\rho \quad \text{for all } i\text{'s}$$

Inserting the values from (C.13) and (C.15) we get:

$$U_i \leq i+k-2Mk' \tag{C.18}$$

Also for the lower bound the following inequality is always satisfied:

$$(P_j)_{min}^{p_{j-1}=i} \geq L_{-\rho} \quad \text{for all } i\text{'s}$$

Using (C.14) and (C.16) we get:

$$L_i \geq i-k+2Mk' \tag{C.19}$$

In order to have maximum overlap between the adjacent regions i and (i+1), U_i should be as large as possible and L_i as small as possible. Therefore, from (C.18) and (C.19) we have:

$$\begin{cases} U_i = i+k-2Mk' \\ L_i = i-k+2Mk' \end{cases} \tag{C.20}$$

and therefore:

$$i+k-2Mk' \geq (P_j)^{p_j=i} \geq i-k+2Mk' \tag{C.21}$$

In order to have overlap between the adjacent regions the following inequality should be satisfied:

$$U_i \geq L_{i+1}$$

from (C.20) we get:

$$M \leq \frac{2k-1}{4k'} \qquad (C.22)$$

This means, the maximum allowable values of the multiplicand and multiplier is equal to $\frac{2k-1}{4k'}$. If the operands are larger than this bound, then there will be a "gap" between adjacent regions. That is, there will be some values of P_j in which there is no acceptable product digit p_j.

For example when r=2 and k=k'=1

$$M \leq \frac{1}{4}$$

So shifting the operands two bits to the right will guarantee the convergence of the algorithm.

Letting j=m in Eq. (C.21) we get:

$$p_m + k - 2Mk' \geq P_m \geq p_m - k + 2Mk'$$

or

$$k - 2Mk' \geq P_m - p_m \geq -k + 2Mk'$$

and since M>0 and k'>0 then Eq. (C.11) is satisfied and R is indeed the product of X and Y to m digits of precision.

REMARKS

From Eq. (C.22) it is clear that the on-line multiplication is not possible when k=1/2 or when the product is not redundant.

APPENDIX D

ON-LINE ADDITION/SUBTRACTION

In this appendix a systematic derivation of on-line addition/subtraction algorithms will be presented. This method is compatible with the methods given for on-line multiplication and division in Appendix C and [GOR 80] respectively. The derivation is applicable to both redundant and non-redundant operands. But, in what follows we only consider addition and subtraction with redundant operands.

Addition

Let the radix r representation of addend, augend and sum be denoted by A, B and R respectively such that:

$$A = \sum_{i=1}^{m} a_i r^{-i} \tag{D.1}$$

$$B = \sum_{i=1}^{m} b_i r^{-i} \tag{D.2}$$

$$R = \sum_{i=0}^{m} s_i r^{-i} \tag{D.3}$$

and R=A+B to m digits of precision.

We assume a_i, b_i, and s_i belong to three different redundant digit sets:

$$a_i \leftarrow \{-\rho', \ldots, \bar{1}, 0, 1, \ldots, \rho'\} \quad (r-1) \geq \rho' \geq r/2 \qquad (D.4)$$

$$b_i \leftarrow \{-\rho'', \ldots, \bar{1}, 0, 1, \ldots, \rho''\} \quad (r-1) \geq \rho'' \geq r/2 \qquad (D.5)$$

$$s_i \leftarrow \{-\rho, \ldots, \bar{1}, 0, 1, \ldots, \rho\} \quad (r-1) \geq \rho \geq r/2 \qquad (D.6)$$

Redundancy coefficients of A,B and R are defined as:

$$\begin{cases} k' = \dfrac{\rho'}{r-1} \\ k'' = \dfrac{\rho''}{r-1} \\ k = \dfrac{\rho}{r-1} \end{cases} \qquad (D.7)$$

The algorithm of the next page generates the sum of A and B in on-line mode. We call this algorithm "ADD". This algorithm is a modification of the addition algorithm shown in [IRW 77].

Algorithm ADD

Step 1 [Initialization]:

$P_0=0$

$s_{-1}=0$

$R_{-1}=0$

For j=1,2,.....,m+1 Do:

Step 2 [Input Digit]:

a_j and b_j

step 3 [Basic Recursion]:

$$P_j=r(P_{j-1}-s_{j-2})+(a_j+b_j)r^{-1} \qquad (D.8)$$

Step 4 [Selection]:

$s_{j-1}=\text{SELECT}(r\hat{P}_j,\hat{c}_j)$

$R_j=R_{j-2}+s_{j-1}r^{-(j-1)}$

Step 5 [End Do]

Proof of Convergence

Inserting different values of j into the basic recursion formula (Eq. D.8) we get:

$$j=1 \rightarrow P_1=(a_1+b_1)r^{-1}$$

$$j=2 \rightarrow P_2=(a_1+b_1)+(a_2+b_2)r^{-1}-rs_0$$

$$j=3 \rightarrow P_3=r(a_1+b_1)+(a_2+b_2)+(a_3+b_3)r^{-1}$$

$$-r^2s_0-rs_1$$

Therefore, P_j is:

$$P_j=r^{j-1}\left[(a_1+b_1)r^{-1}+(a_2+b_2)r^{-2}+\ldots+(a_j+b_j)r^{-j}\right]$$

$$-r^{j-1}\left[s_0+r^{-1}s_1+\ldots+r^{-(j-2)}s_{j-2}\right]$$

or

$$P_j=r^{j-1}\sum_{i=1}^{j}(a_i+b_i)r^{-i}-r^{j-1}\sum_{i=0}^{j-2}s_ir^{-i} \qquad (D.9)$$

if j=m+1, then:

$$P_{m+1}=r^m\sum_{i=1}^{m}(a_i+b_i)r^{-i}-r^m\sum_{i=0}^{m-1}s_ir^{-i} \qquad (D.10)$$

Using (D.3) we get:

$$R=\sum_{i=0}^{m-1}s_ir^{-i}+s_mr^{-m}$$

or

$$\sum_{i=0}^{m-1} s_i r^{-i} = R - s_m r^{-m}$$

inserting this into (D.10) we obtain:

$$P_{m+1} = r^m(A+B) - r^m(R - s_m r^{-m})$$

rearranging the terms we get:

$$R = (A+B) - r^{-m}(P_{m+1} - s_m) \qquad (D.11)$$

By devising a sum digit selection procedure, SELECT, in step 4 of the Algorithm "ADD", such that:

$$|P_{m+1} - s_m| \leq k \qquad (D.12)$$

R=A+B can be computed to m digits of precision.

Selection Rules

Figure (D.1) shows a selection graph for the operation of addition. This is a plot of shifted partial sum versus the sum of the operand digits (a_j and b_j) at step j. We designate this as P-c plot. The notations used in this graph are similar to those used in Appendix C.

In order to derive the range restriction on P_j, our only assumption will be the basic recursion formula (D.8) and the fact that P_j should bounded.

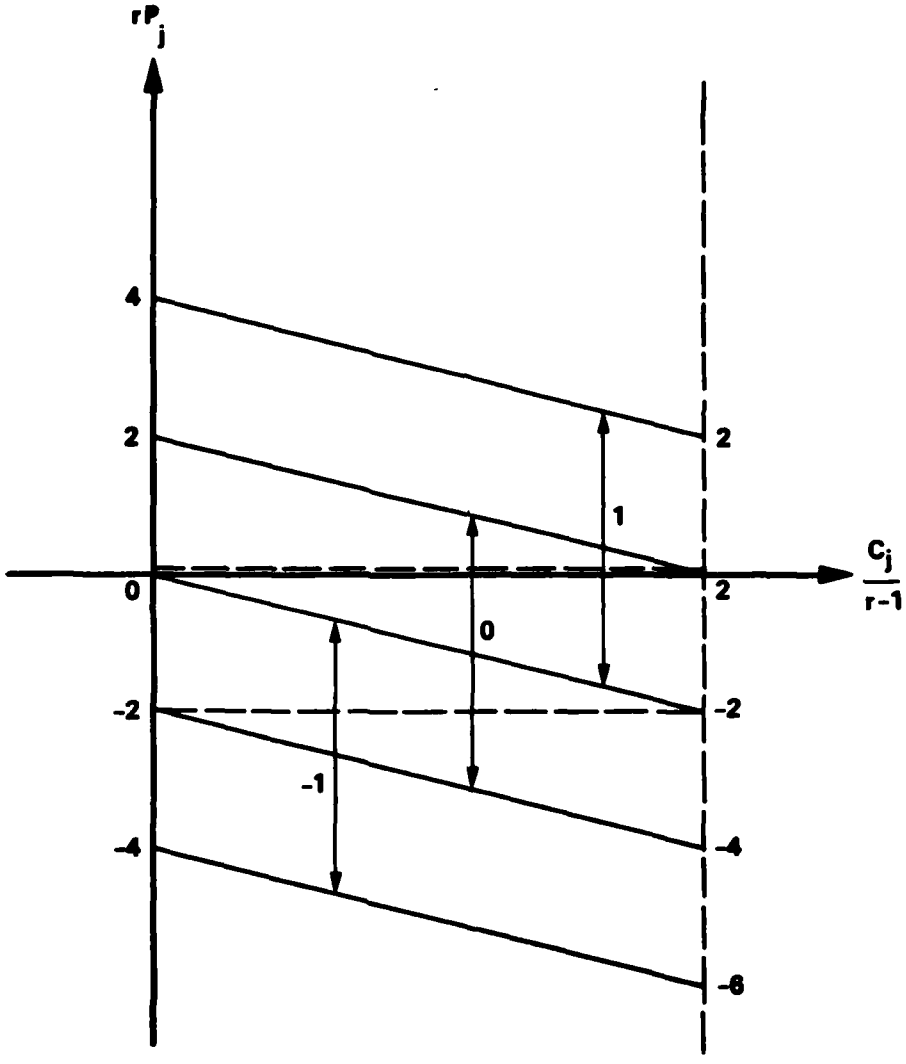


Figure D.1 – A P-C Plot

$$\begin{cases} P_j = r(P_{j-1} - s_{j-2}) + c_j r^{-1} \\ c_j = a_j + b_j \end{cases} \tag{D.13}$$

The maximum and the minimum values of P_j are obtained as follows:

$$(P_j)_{max}^{s_{j-2}=i} = U_i - ri + c_j r^{-1} \tag{D.14}$$

$$(P_j)_{min}^{s_{j-2}=i} = L_i - ri + c_j r^{-1} \tag{D.15}$$

when $s_{j-2}=\rho$ from (D.14) we get:

$$(P_j)_{max}^{s_{j-2}=\rho} = U_\rho - r\rho + c_j r^{-1}$$

since P_j should be bounded, this value should be equal to $\frac{U_\rho}{r}$, therefore:

$$U_\rho - r\rho + c_j r^{-1} = \frac{U_\rho}{r}$$

or

$$U_\rho = r^2 k - \frac{c_j}{r-1} \tag{D.16}$$

Similar to this, for the lower bound we obtain:

$$(P_j)_{min}^{s_{j-2}=-\rho} = L_{-\rho} + r\rho + c_j r^{-1} = \frac{L_{-\rho}}{r}$$

This results in:

$$L_{-\rho} = -r^2 k - \frac{c_j}{r-1} \tag{D.17}$$

from (D.16) and (D.17), (D.18) will be obtained:

$$r^2k - \frac{c_j}{r-1} \geq rP_j \geq -r^2k - \frac{c_j}{r-1} \qquad (D.18)$$

Thus, the upper and lower bounds of rP_j are implied by the recursion formula and not by the selection procedure (as shown in [IRW 77]).

Selection regions are obtained using Equations (D.14), (D.15), (D.16) and (D.17) as follows:

$$(P_j)_{max}^{s_{j-2}=i} \leq \frac{U\rho}{r} \quad \text{for all i's}$$

plugging Equations (D.14) and (D.16) in the above inequality we get:

$$U_i \leq r(i+k) - \frac{c_j}{r-1} \qquad (D.19)$$

Similarly from Equations (D.15) and (D.17) and the relation:

$$(P_j)_{min}^{s_{j-2}=i} \geq \frac{L\rho}{r} \quad \text{for all i's}$$

we get:

$$L_i \geq r(i-k) - \frac{c_j}{r-1} \qquad (D.20)$$

In order to have maximum overlap between the adjacent regions of the P-c plot, the equality signs of Equations (D.19) and (D.20) should be satisfied. Therefore:

$$U_i = r(i+k) - \frac{c_j}{r-1} \qquad (D.21)$$

$$L_i = r(i-k) - \frac{c_j}{r-1} \qquad (D.22)$$

thus, the selection regions are specified by the following inequality:

$$r(i+k) - \frac{c_j}{r-1} \geq rP_j^i \geq r(i-k) - \frac{c_j}{r-1} \qquad (D.23)$$

In order to have overlaps between the adjacent regions of the P-c plot, U_i should be greater than L_{i+1} for all i's. That is:

$$U_i \geq L_{i+1} \rightarrow k \geq \frac{1}{2}$$

Therefore, there is always overlap between the adjacent regions of the P-c plot.

In order to prove that the relation (D.12) is satisfied by the given selection procedure, we rewrite Eq. (D.23) as:

$$r(s_{j-1}+k) - \frac{c_j}{r-1} \geq rP_j \geq r(s_{j-1}-k) - \frac{c_j}{r-1}$$

or

$$k - \frac{c_j}{r(r-1)} \geq P_j - s_{j-1} \geq -k - \frac{c_j}{r(r-1)}$$

when j=m+1 we obtain:

$$k - \frac{c_{m+1}}{r(r-1)} \geq P_{m+1} - s_m \geq -k - \frac{c_{m+1}}{r(r-1)}$$

since $a_{m+1}=b_{m+1}=0 \rightarrow c_{m+1}=0$ and therefore:

$$k \geq P_{m+1} - s_m \geq -k$$

or

$$|P_{m+1} - s_m| \leq k$$

so condition (D.12) is satisfied by the given selection procedure and R is indeed the sum of A and B to m digits of precision.

Subtraction

Since the subtrahend is represented in redundant format, subtraction can be performed by just flipping the sign of the subtrahend digits and following the addition procedure given above.

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER UCLA-ENG-8197	2. GOVT ACCESSION NO. AD-A098	3. RECIPIENT'S CATALOG NUMBER 886
4. TITLE (and Subtitle) ERROR-CODED ALGORITHMS FOR ON-LINE ARITHMETIC		5. TYPE OF REPORT & PERIOD COVERED FINAL 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) ABDOLALI GORJI-SINAKI		8. CONTRACT OR GRANT NUMBER(s) N00014-79-C-0866
9. PERFORMING ORGANIZATION NAME AND ADDRESS University of California Los Angeles Computer Science Department Los Angeles, California 90024		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS NRSR0-008
11. CONTROLLING OFFICE NAME AND ADDRESS Office of Naval Research 1030 East Green Street Pasadena, California 91106		12. REPORT DATE FEBRUARY 1981 13. NUMBER OF PAGES
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) Office of Naval Research 1030 East Green Street Pasadena, California 91106		15. SECURITY CLASS. (of this report) Unclassified 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

1. Defense Documentation Center, VA
2. Naval Research Laboratory, D.C.
3. Dr. Slafkosky-Scientific Advisor, D.C.
4. Mr. Gleissner-Naval Ship Research and Development Computation & Mathmetics Dept., MD
5. Captain Grace Hopper (008) Naval Data Automation Command Washington, D.C.
6. Offices of Naval Research in VA, MA, IL, CA, and D.C.

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

On-Line Arithmetic, Error Codes, Fault-Tolerance, Multi-Module Implementation

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Since on-line arithmetic requires relatively long sequences of operations in order to achieve speed-up over conventional arithmetic, it is important to protect on-line algorithms against hardware failures. If not protected, the hardware failures could quickly contaminate large number of results in progress due to tight coupling of the steps at the digit level. By detecting errors, as they occur, an effective, gracefully degradable organization could be achieved. Namely, error at any step of the algorithms would lead to restriction of precision (significance) of the remaining steps but not catastrophic termination.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 68 IS OBSOLETE
S/N 0102-LF-014-6601

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

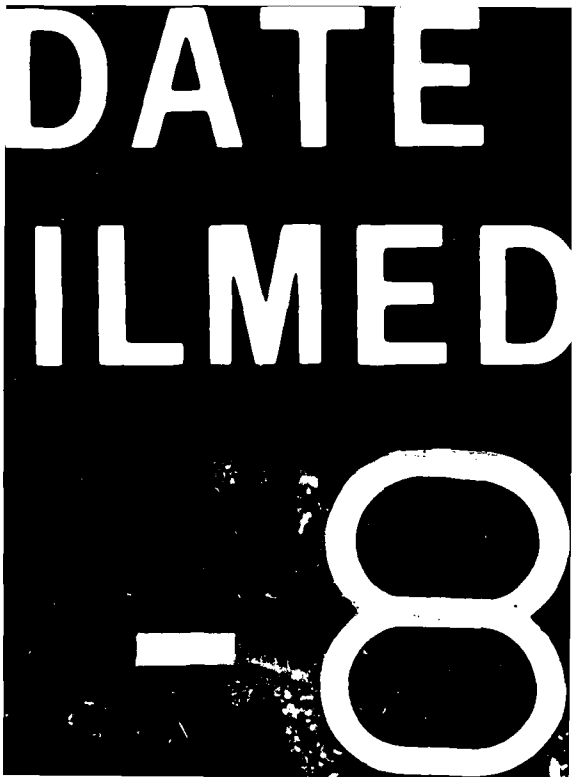
cont.

The main objective of this dissertation is to develop and demonstrate the feasibility of error-coded on-line arithmetic suitable for distributed systems.

In this thesis a set of error-coded on-line algorithms was developed for the four basic operations of addition/subtraction, multiplication and division. Low cost arithmetic error codes (Residue and AN Codes) were found to be suitable for this purpose.

Hardware design of the error-coded units at the gate level was considered. A residue-coded on-line division unit was designed based on a already designed digit-slice division unit.

A general mathematical model for the cost and speed of the error-coded units was derived and was compared with similar values when no error code is used. Finally, the effectiveness of the proposed detection/correction schemes was considered and proved.


DATE
ILMED
-8